

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

based on
European Multilaterally Secure Computing Base (EMSCB)



Abstract: The aim of this EMSCB-based security service is to provide a strongly isolated hard-disk 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	Introduction	3
2	Security Environment	4
2.1	Assumptions	4
/A 10/	Correct hardware	4
/A 20/	No man-in-the-middle attack	5
/A 30/	Trusted Administrator	5
2.2	Threats	5
/T 10/	Key spoofing	5
/T 20/	Spoofing of authentication information	5
/T 30/	Key manipulation	5
/T 40/	Faked user interface	5
/T 50/	Faked identity	5
/T 60/	Software manipulation	5
/T 70/	Device driver manipulation	5
3	Functional Requirements (Use Case Model)	6
3.1	Goal	6
3.2	Target Groups	6
3.3	Overview	6
3.4	Roles and Actors	9
3.5	Use Cases	9
/UC 10/	User Authentication	9
/UC 20/	Register Encrypted Resources	10
/UC 30/	Deregister Encrypted Resources	11
/UC 40/	Password Change	11
/UC 50/	Initialization	12
/UC 51/	Add User Account	13
/UC 52/	Delete User Account	13
/UC 53/	User Rights Administration	14
/UC 54/	Add Resource	15
/UC 55/	Delete Resource	15
/UC 60/	Smart Card Sign-On	16
/UC 70/	Smart Card Change	17
4	Security Objectives	17
/SO 10/	Confidentiality of the cryptographic key	17
/SO 20/	Integrity of the cryptographic key	18
/SO 30/	No unauthorized use of the cryptographic key	18

5	Supplementary Requirements	18
5.1	Preconditions	18
	/PR 110/ Microkernel	18
	/PR 120/ L4env	18
	/PR 130/ L4Linux	18
5.2	Required Criteria	18
	/FR 110/ dm_crypt	18
	/FR 120/ Virtual Block Device	18
	/FR 130/ AES	18
	/FR 140/ Secure GUI	18
	/FR 150/ Single-User Support	19
	/FR 170/ Password Change	19
	/FR 180/ DMA Disabling	19
5.3	Desired Criteria	19
	/FD 210/ Multi-User Support	19
	/FD 215/ Persistent Storage	19
	/FD 220/ TPM Support	19
	/FD 230/ Smart Card Support	19
	/FD 240/ Blowfish	19
5.4	Distinguishing Criteria	19
5.5	Execution Environment	19
	5.5.1 Software	19
	5.5.2 Hardware	20
5.6	Development Environment	20
	5.6.1 Software	20
	5.6.2 Hardware	20
	5.6.3 Development Interface Requirements	20
5.7	Product Data	20
	/D 10/ Cryptographic Keys	20
	/D 20/ Identities (optional)	20
5.8	Graphical User Interface (GUI)	21
6	Security Requirements	21
	/SR 10/ User Authentication	21
	/SR 20/ Application Authentication	21
	/SR 30/ Isolation	21
	/SR 40/ Secure Communication	21
	/SR 50/ Trusted Path	21
	/SR 60/ Trusted Hardware Access	21
7	References	22

1 Introduction

Encrypting the data stored on a hard-disk is a commonly used security technique to protect the confidentiality of the data, especially when used on mobile computer platforms like laptop PCs. There exist several software-based encryption products. Some of them are shipped together with the operating system. One example is the Encrypting File System (EFS) of the Windows operating system. Another example would be Linux and its `dm_crypt`, which allows different encryption algorithms to plug in into the kernel and use them for encrypting file systems.

Most software-based hard-disk encryption products suffer on insecure storage and usage capabilities for security-critical cryptographic keys and operations. The used operating systems (OS) that control all data storage mechanisms, i.e. hard-disk, memory, USB, I/O etc., cannot definitely assure that other (probably malicious) applications cannot gain access to the critical key data. This can be seen by the huge number of exploits and continuous security updates due to various conceptual weaknesses, e.g. the monolithic OS kernel architecture and thus the increased complexity. This pertains Windows-based operating systems as well as Linux-based ones. The main reason is due to the fact that a large part of the operating system and supporting processes are executed in a privileged mode, the so called kernel mode, which allows them to directly access the hardware and all other software processes. User applications are usually executed in a non-privileged mode, the so called user mode.

Thus, the risk of security weaknesses is higher because of the huge amount of code executed in privileged mode. If such a process can be exploited it is possible to gain access to all kernel data, including the encryption keys used for the hard-disk encryption.

Most important, even if it would be possible to build a safe and secure OS, it would be impossible to store critical key material securely, since booting the computing platform into another potentially malicious OS, e.g., from the floppy disk, always circumvents all enforcement mechanisms.

Therefore, all security-critical cryptographic keys and operations should be handled by a ‘secure compartment’, that itself is fully independent, strongly protected and isolated [7] from the standard (Linux or Windows) OS. This requires an underlying mechanism which is capable of isolating the normal operating system from particular applications. Thus, the tamper-resistance assumption of the normal operating system could be reduced.

After a successful authentication process against the secure compartment, the OS mechanism that handles the encryption process just sends the plain text to the isolated compartment and receives the cipher text afterwards and vice versa without having access to any cryptographic key data.

The authentication process could simply authenticate a qualified user, i.e. the data owner, and then provide access to the data to all applications of the respective user. Beyond that, it is also possible to associate data access with certain security policies, such that access to particular data is only provided if the computing platform fulfills the required policies.

Trusted Computing technology, as provided by the Trusted Platform Module(TPM),

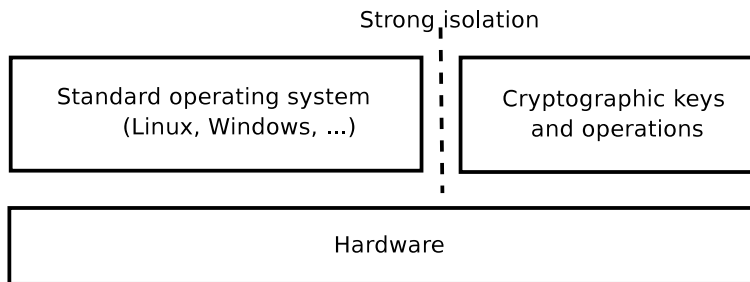


Figure 1: Isolated cryptographic keys and operations

can be used and integrated to enforce such policies. Data like the cryptographic keys used for encryption can be *sealed* to a certain platform configuration. If the platform is in a different configuration, the encryption keys cannot be accessed and hence the confidential data cannot be decrypted.

Moreover, the TPM can be used to detect modified operating system modules when the system is started. This can be used to prevent the execution of malicious code and to establish a secure booting mechanism.

The objective target of TPM support is to integrate the following mechanisms:

- Secure booting
- Sealing

The protected encryption compartment prevents any other Linux application from accessing the secret key information. Therefore, it clearly enhances the security of common software-based encryption schemes and provides full usability while implementing the standard Linux devices encryption interface. However, primarily it clearly demonstrates the particular security enhancing capabilities of EMSCB. Furthermore, the optional use of Trusted Computing technology can leverage the security of the whole system by restricting the access to the encryption keys to certain platform configurations.

2 Security Environment

This section describes the security aspects of the environment in which the product is intended to be used and the manner in which it is expected to be employed.

2.1 Assumptions

/A 10/ Correct hardware

The underlying hardware (e.g., CPU, devices, TPM, ...) is non-malicious and behaves as specified.

/A 20/ No man-in-the-middle attack

An attack using a dummy device that relays the whole communication between the user and the platform to another device does not happen.

/A 30/ Trusted Administrator

The compartment administrator of the system must be trusted since he will have access to all encrypted data.

2.2 Threats

/T 10/ Key spoofing

An adversary may try to eavesdrop the cryptographic key used for encryption/decryption.

/T 20/ Spoofing of authentication information

An adversary may try to eavesdrop the user authentication information.

/T 30/ Key manipulation

An adversary may try to violate integrity requirements of the cryptographic key used for encryption/decryption.

/T 40/ Faked user interface

An adversary may try to deceive users by a platform providing a faked user interface.

/T 50/ Faked identity

An adversary may try to bypass control mechanisms by pretending a faked identity.

/T 60/ Software manipulation

An adversary may try to violate security requirements by maliciously manipulating the security kernel.

/T 70/ Device driver manipulation

An adversary may try to manipulate device drivers such that hardware functions (e.g., direct memory access) are used to violate security policies.

3 Functional Requirements (Use Case Model)

3.1 Goal

The aim of this EMSCB-based security service is to provide a strongly isolated hard-disk 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.

The goal of this system is to protect confidentiality of data against unauthorized usage, especially in the case of thievery and physical attack, i.e., direct electromagnetic measuring methods on the hard-disk platters. In detail, the goal is to protect the confidentiality and integrity of the cryptographic keys used for encryption.

However, we do not address the case when a user application that is allowed to access the decrypted information is going to send it to an unauthorized user or application. It is not the goal of this system to protect against attacks aiming at redirecting the information flow.

3.2 Target Groups

- Notebooks (single user scenario)
- Desktop systems (multi-user scenario)

3.3 Overview

The EMSCB HDD-Encrypted will support three different user modes:

- **Mode 1:** Single-user mode without GUI
- **Mode 2:** Single-user mode with GUI
- **Mode 3:** Multi-user mode (always with GUI)

In the single-user mode there is only one user. The user uses one single key to encrypt one or more resources. The user has to provide a password, which will be used to derive an encryption key. The user's password can be provided by a graphical user interface dialog or by the bootloader at the time of loading the encryption compartment.

Figure 2 shows the uses cases involved in the single-user scenario. In this case there is no extra graphical user interface for entering a password. Instead, the user is asked at boot time (e.g., via GRUB bootloader) for a password. This password is then passed to the protected encryption compartment. When the user wants to register, i.e. mount, an encrypted resource the password provided by at boottime will be used to derive the encryption key. Furthermore, the user can deregister, i.e. unmount, his encrypted resources.

Figure 3 shows the uses cases of the single-user scenario if a special GUI dialog shall be provided to enter the password. Again, the user can register and deregister encrypted resources. This time, however, the user has to authenticate himself at the time of registering the first encrypted resources. A special dialog using a secure GUI

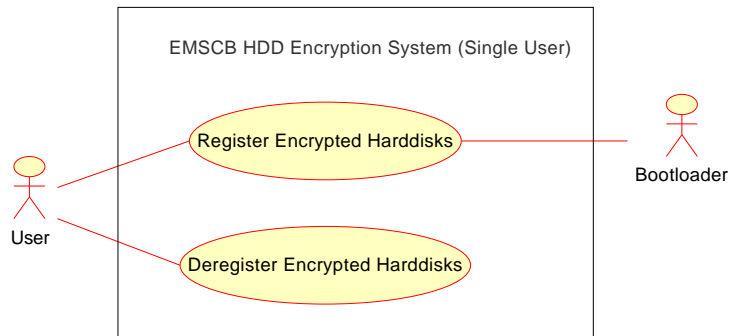


Figure 2: Resource encryption in single-user mode without GUI

will ask the user to enter his password. In contrast to the first scenario the user will have to authenticate himself again when he registers a resource after all resources have been deregistered.

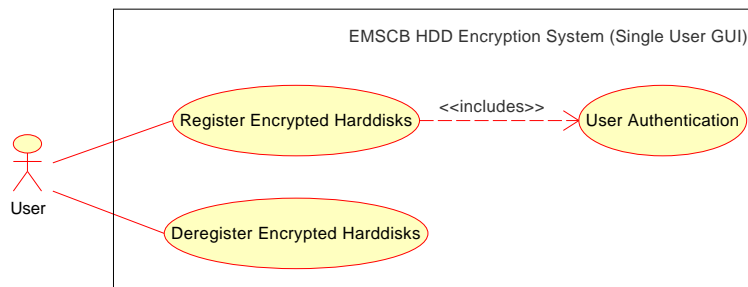


Figure 3: Resource encryption in single-user mode with GUI

In the multi-user mode there can be several users with several encrypted resources. It is possible to share encrypted resources between two or more users. Moreover, at the same time a user can have distinct encrypted resources that cannot be accessed, i.e. decrypted, by other users. The uses cases in this mode are shown in Figure 4. Again, the user can register and deregister encrypted resources. However, an encrypted resource can be shared by several users. This requires to share the secret encryption key as well. But those users should be able to have different authorization data, e.g. passwords. For that purpose there is a compartment administrator, who assigns access rights of encrypted resources to the different users. This requires new use cases which pertain to user and resource management.

The compartment administrator can add or delete user accounts, add or delete resources (which identify the individual encryption keys to be used) and assign or modify user rights that are necessary to access encrypted resources. When the compartment is started for the very first time the administrator has to initialize the system's configuration database. Most of these use cases require an authentication of the user or

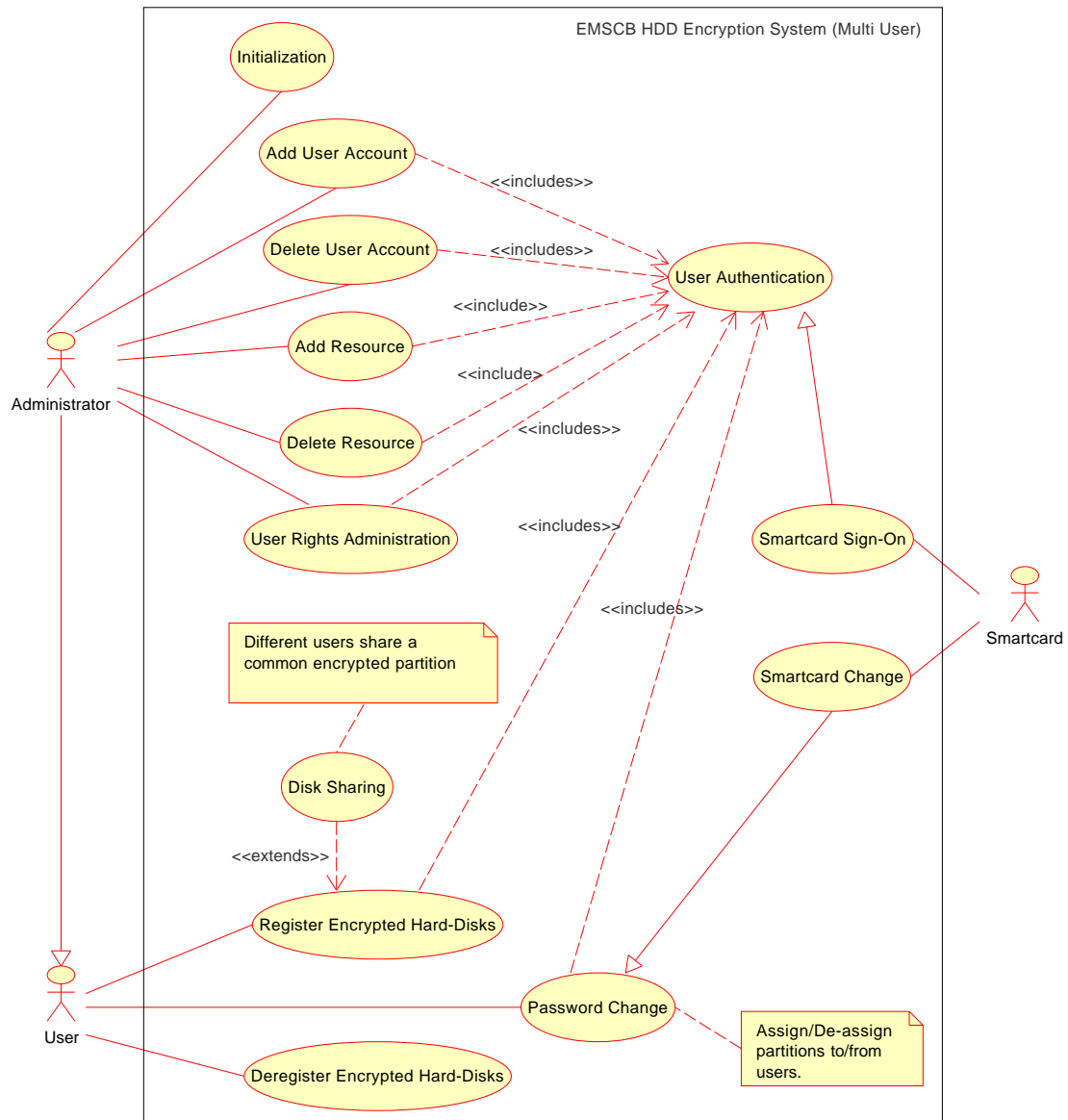


Figure 4: Resource encryption in multi-user mode

administrator to authorize his request. The authentication process can be performed by asking for a password or, optionally, by providing a smart card. The users should be able to change their password or smart cards.

3.4 Roles and Actors

User A user can register, access and deregister encrypted resources that belong to him. Moreover he can change his respective authorization data, but a user cannot modify other user accounts or change the configuration of the encryption compartment.

Administrator An administrator can modify the configuration of the encryption compartment: he can create and delete user accounts, grant and revoke access rights for resources and grant and revoke the administration right to user accounts. Moreover the administrator can create the initial configuration of the encryption compartment. The administrator can access all encrypted resources, i.e. he has access to the plain text information.

Bootloader The bootloader loads the operating system and the encryption compartment into the memory and starts the system. In the single-user scenario without GUI the bootloader is also responsible for asking the user a password and passing this password to the encryption compartment.

Smart card A smart card is an external system used to authenticate the user. Usually it uses a cryptographic public-key protocol to perform the authentication. In general, the smart card can be any secure token which is able to allow an authentication by possession. In our model, any devices needed to connect the token to the system are included in the smart card actor.

3.5 Use Cases

/UC 10/ User Authentication

Description The current user authenticates himself with the corresponding password.

Actors User, Administrator

Rationale Prove user's identity and access authorization.

Normal Flow

1. The system asks the user for authentication data (password).
2. The user enters the password into the password dialog of the secure GUI.
3. The system authorizes the user.

Alternative Flow

1. The system asks the user for authentication data (password).
2. The user enters a wrong password into the password dialog of the secure GUI.
3. The system refuses authorization.

Extensions Within a multi-user environment, the system also asks for the corresponding user name.

/UC 20/ Register Encrypted Resources

Description After a successful authentication, the resources that belong to the user will be mounted and decrypted automatically.

Actors User

Includes /UC 10/ User authentication

Rationale Access encrypted resources.

Preconditions The encrypted resources have not already been registered before.

Normal Flow

1. The user requests access to an encrypted resource.
2. The system asks for authorization of his intention.
3. The user verifies his access right using /UC 10/ User authentication.
4. After successful authorization all encrypted resources that belong to the user will be encrypted/decrypted and mounted automatically.

Alternative Flow

1. The user requests access to an encrypted resource.
2. The system asks for authorization of his intention.
3. The user fails the authentication for access using /UC 10/ User authentication
4. The system refuses access to requested encrypted resource.

Alternative Flow (Single User)

1. The user requests access to an encrypted resource.
2. The system asks the bootloader for authorization data.
3. After successful authorization all encrypted resources will be encrypted/decrypted and mounted automatically.

Extensions Within a multi-user environment, the automatic mounting process includes all shared group resources and also all resources the user shares together with others.

/UC 30/ Deregister Encrypted Resources

Description A registered user deregisters his access to encrypted resources that belong to him.

Actors User

Rationale Sign off from all encrypted resources including shared resources so that the plain-text information cannot be accessed any more (until the next registering).

Preconditions User has already registered.

Postconditions All encrypted resources of the user are deregistered and the contained information will not be automatically decrypted until the next registering.

Normal Flow

1. The user requests to deregister his encrypted resources.
2. The system asks for acknowledgment of his intention.
3. After the positive acknowledgment the system will deregister the user from all encrypted resources that belong to the user.

Extensions Within a multi-user environment, the automatic demounting process includes all shared group resources and also all resources the user shares together with others.

/UC 40/ Password Change

Description After successful verification of his current authorization data, the user can renew/change his authentication data (password). The user is not able to change his username.

Actors User, Administrator

Rationale User wants to renew its authentication data.

Includes /UC 10/ User authentication

Preconditions User has already registered.

Postconditions After successful accomplishment the user's authentication data is changed.

Normal Flow

1. The user requests to change his authentication data.
2. The system asks for authorization of his intention.
3. The user proves his authorization using /UC 10/ User authentication.
4. After successful authorization the user can renew his authentication data (password).

Alternative Flow

1. The user requests to change his authentication data.
2. The system asks for authorization of his intention.
3. The user fails the authentication using /UC 10/ User authentication.
4. System refuses changing his authentication data.

Alternative Flow

1. The user requests to change his authentication data.
2. The system asks for authorization of his intention.
3. The user cancels the operation.
4. The system aborts the renew process.

Notes Instead of the user, the administrator can perform the same steps. To avoid accidentally mistyped authorization data, it is conceivable that the user has to enter his new authorization data twice.

/UC 50/ Initialization

Description The system administrator can set-up the initial administrator account.

Actors Administrator.

Rationale

- Obligatory for /UC 51/ Add User Account, /UC 52/ Delete User Account, /UC 53/ User Rights Administration, /UC 54/ Add Resource, and /UC 55/ Delete Resource to initially identify the administrator.
- /UC 50/ through /UC 55/ are convenience for /UC 20/ Register encrypted resources; otherwise only predefined users could use the system.

Preconditions The configuration data for the system is not existent or invalid.

Postconditions The configuration data is valid and contains one user who is an administrator.

Normal Flow

1. The following inputs are requested:
 - (a) Username
 - (b) Password
2. The administrator enters the requested information or cancels the operation.
3. A new empty configuration database is created and the new user is added to the configuration database. The user is marked as an administrator.
4. A confirmation message is displayed.

/UC 51/ Add User Account

Description The system administrator can add user accounts. Management is possible by using a graphical user interface.

Actors Administrator.

Rationale Convenience for /UC 20/ Register Encrypted Resources; otherwise only predefined users could use the system.

Includes /UC 10/ User Authentication

Preconditions

- The system has been initialized, see /UC 50/.
- The system administrator has logged in with a user account marked as an administrator.
- The configuration exists, has been loaded and is valid.

Normal Flow

1. The following inputs are requested:
 - (a) Username
 - (b) Initial password
2. The administrator enters the requested information or cancels the operation.
3. The given username is checked against the existing usernames for duplicates.
 - If the given username is unique, the new user is added to the configuration database without any access rights and a confirmation message will displayed.
 - If the given username is not unique, a message is displayed and the administrator can alter the requested information.

/UC 52/ Delete User Account

Description The system administrator can delete user accounts. Management is possible by using a graphical user interface.

Actors Administrator.

Rationale Convenience for /UC 20/ Register Encrypted Resources; otherwise only predefined users could use the system.

Includes /UC 10/ User Authentication

Preconditions

- The system has been initialized, see /UC 50/.
- The system administrator has logged in with a user account marked as an administrator.
- The configuration exists, has been loaded and is valid.

Normal flow

1. The administrator selects the user account that shall be deleted or cancels the operation.
 - The administrator cannot delete his own user account. If the selected user account is the administrator's account, a message is displayed and another account can be selected.
 - Otherwise a warning message is displayed that must be accepted or the operation can be canceled.
2. If the warning message is accepted, the selected user account and all its access rights are deleted from the configuration database.

/UC 53/ User Rights Administration

Description The system administrator can grant and revoke access rights for user accounts. Management is possible by using a graphical user interface.

Actors Administrator.

Rationale Convenience for /UC 20/ Register Encrypted Resources; otherwise only predefined users could use the system.

Includes

- /UC 10/ User Authentication

Preconditions

- The system has been initialized, see /UC 50/.
- The system administrator has logged in with an user account marked as an administrator.
- The configuration exists, has been loaded and is valid.

Normal flow

1. The administrator selects the user account that shall be altered.
2. A listing of the access rights of the selected user account is being displayed.
3. The administrator selects a resource for which access rights shall be changed *OR* the administrator selects the „administration flag“
4. The administrator enters the new access right (allowed / not allowed) for the selected resource or administration flag.
5. The administrator confirms the desired changes.
 - The administrator cannot revoke the „administration flag“ for his own account: If the selected user account is the administrator's account and if the „administration flag“ shall be set to „not allowed“ an error message is displayed

- Otherwise the new access rights are stored in the configuration database.
6. A message showing the new access rights is displayed.

/UC 54/ Add Resource

Description The system administrator can add resources (e.g. hard-disk partitions) that can be accessed by users. Management is possible by using a graphical user interface.

Actors Administrator.

Rationale Convenience for /UC 20/ Register Encrypted Resources; otherwise only predefined resources could be used.

Includes /UC 10/ User Authentication

Preconditions

- The system has been initialized, see /UC 50/.
- The system administrator has logged in with a user account marked as an administrator.
- The configuration exists, has been loaded and is valid.

Normal Flow

1. The system asks for the following input:
Name of the resource (e.g. Linux partition names like /dev/sda1)
2. The administrator enters the requested information
3. The system checks the given resource name against the existing resources for duplicates.
 - If the given resource is unique, the system adds the new resource to the configuration database. A new cryptographic key for this resource is being generated. A confirmation message is displayed.
 - If the given resource is not unique, the system displays a message and the administrator can alter the requested information.

Alternative Flow

The administrator cancels the operation in step 2.

Alternative Flow (TPM Support)

In step 2 the administrator can optionally choose to bind the cryptographic key for the new resource to the specific hardware platform and/or to the encryption compartment.

/UC 55/ Delete Resource

Description The system administrator can delete resources. Management is possible by using a graphical user interface.

Actors Administrator.

Rationale Convenience for /UC 20/ Register Encrypted Resources; otherwise only predefined resources could be used.

Includes /UC 10/ User Authentication

Preconditions

- The system has been initialized, see /UC 50/.
- The system administrator has logged in with a user account marked as an administrator.
- The configuration exists, has been loaded and is valid.

Normal Flow

1. The administrator selects the resource that shall be deleted.
2. The system displays a warning message that shows all user accounts that have access rights for the selected resource.
3. The administrator accepts the warning message.
4. The system deletes the selected resource and all access rights of users to this resource in the configuration database.

Alternative Flow

The administrator can cancel the operation in steps 1 or 3 above.

/UC 60/ Smart Card Sign-On

Description The user wants to sign-on the system and mount his encrypted disks using a smart card (more generally: a secure token) to identify and authenticate himself.

Actors User, Smart card.

Specializes /UC 10/ User Authentication

Preconditions The system possesses information to identify and authenticate the user. Furthermore, the system possesses information to decrypt the user's disk or can derive this information.

Postconditions The System decrypts the user's disks transparently from now on until the user deregisters his encrypted disks. The user can work with the data on his disks without any further sign-on procedure to decrypt the data.

Normal Flow

1. The system asks the user for identification and authentication.
2. The user inserts his smart card (secure token) into the card reader.
3. The smart card authenticates the user.
4. The system authorizes the user to login and mounts his encrypted disks.

Alternative Flow

1. The system asks the user for identification and authentication.
2. The user inserts his smart card (secure token) into the card reader.
3. The authentication fails, the system aborts the procedure.

/UC 70/ Smart Card Change

Description The user wants to change his smart card or the key on it. Data encrypted with the old smart card has to be re-encrypted with the new one.

Actors User, Smart card.

Specializes /UC 40/ Password Change

Preconditions The user and his old smart card are registered at the system. The user's disks are encrypted with key information provided by the old smart card.

Postconditions The new smart card is registered at the system. The user's disks are encrypted with key information provided by the new smart card.

Normal Flow

1. The user identifies and authenticates himself providing the old smart card.
2. The system authorizes the user to change his smart card.
3. The user selects the new smart card and approves his selection.
4. The system re-encrypts all data encrypted with the old smart card using the new smart card now.

Alternative Flow

1. The user identifies and authenticates himself providing the old smart card.
2. The authentication fails, the system aborts the procedure.

Alternative Flow

1. The user identifies and authenticates himself providing the old smart card.
2. The system authorizes the user to change his smart card.
3. The user selects the new smart card and cancels his selection.
4. The system aborts the procedure.

4 Security Objectives

/SO 10/ Confidentiality of the cryptographic key

Unauthorized entities, must not be able to read the cryptographic key used for encryption/decryption. Entities in this context (optionally) comprise software and/or hardware components as well.

/SO 20/ Integrity of the cryptographic key

Unauthorized entities must not be able to overwrite the cryptographic key used for encryption/decryption.

/SO 30/ No unauthorized use of the cryptographic key

Unauthorized entities must not be able to use the cryptographic key to encrypt or decrypt data.

5 Supplementary Requirements

5.1 Preconditions

/PR 110/ Microkernel

An implementation of the L4V2 μ -kernel API has to be available.

/PR 120/ L4env

Hardened, lightweight L4Env for L4V2 μ -kernel API has to be available.

/PR 130/ L4Linux

L4Linux 2.6 for L4V2 μ -kernel API has to be available.

5.2 Required Criteria

/FR 110/ dm_crypt

Integration of protected encryption compartment into Linux *dm_crypt* module.

/FR 120/ Virtual Block Device

Transparent access to protected encryption compartment for block devices via a respective virtual block device.

/FR 130/ AES

AES block encryption/decryption module with 128-256 Bit keys that provides at least ECB and CBC operation modes.

/FR 140/ Secure GUI

The Secure GUI DOpE (Desktop Operating Environment) as part of L4Env must be used for security-critical user interaction.

/FR 150/ Single-User Support

Single-user support, i.e. 1 password \leftrightarrow n resources.

/FR 170/ Password Change

Password(s) can be changed.

/FR 180/ DMA Disabling

Extraction of all DMA and Busmaster capable devices (i.e. IDE, network and graphics) into isolated compartments and deactivation of all other DMA and bus master capable devices within Linux.

5.3 Desired Criteria

/FD 210/ Multi-User Support

Multi-user support (up to m passwords \leftrightarrow n resources).

/FD 215/ Persistent Storage

Secure Storage is available.

/FD 220/ TPM Support

TPM support to bind the encryption process to certain platform configurations.

/FD 230/ Smart Card Support

Smart card support for user authentication and key migration.

/FD 240/ Blowfish

Additional Blowfish encryption/decryption module.

5.4 Distinguishing Criteria

The isolated encryption compartment is not an independent cryptographic file system (FS), but an additional, external encryption/decryption application for a given cryptographic FS infrastructure.

5.5 Execution Environment

5.5.1 Software

- Fiasco L4V2 μ -kernel [4]
- L4Env 0.2 [5]

- L4Linux 2.6.13 [6]
- L4Linux *cryptsetup* command tool

5.5.2 Hardware

- x86 IBM Thinkpad 41p
- TPM 1.1b (optional)
- GemPlus Smart card (optional)

5.6 Development Environment

5.6.1 Software

- Linux 2.6
- GCC (gcc/g++) 3.3.6
- Fiasco L4V2 μ -kernel [4]
- L4Env 0.2 including DOpE [3]
- VMware 4.5.2
- DICE 2.2.8 or better with C++ backend [2]
- CVS/Aegis

5.6.2 Hardware

IBM x86 Thinkpad 41p.

5.6.3 Development Interface Requirements

- CORBA IDL for DICE, or
- DCE IDL for DICE [1]

5.7 Product Data

/D 10/ Cryptographic Keys

Securely store all critical cryptographic key information isolated from Linux. Optionally, critical key information can be securely stored using the potentially available TPM hardware chip or connected smart cards.

/D 20/ Identities (optional)

Meta-data of user identities and roles have to be managed by the product.

5.8 Graphical User Interface (GUI)

The Secure GUI DOpE (Desktop Operating Environment) as part of L4Env may be used as *Trusted Viewer* for graphical user authentication and user management. The security of this GUI pertains to the fact that Linux processes cannot take over control of or directly exert influence on GUI elements outside of Linux.

6 Security Requirements

/SR 10/ User Authentication

The encryption compartment should be able to verify the correctness of a claimed identity.

/SR 20/ Application Authentication

Users should be able to verify the trustworthiness of the currently active software stack including hardware, security kernel, and application.

/SR 30/ Isolation

Protection of the integrity and confidentiality of both application code and data. This property is required to prevent concurrent processes from directly accessing the cryptographic key or indirectly by manipulating the code of the encryption compartment.

/SR 40/ Secure Communication

Confidentiality, integrity, and authenticity of inter-process communication has to be provided to prevent unauthorized processes from eavesdropping security-critical data sent between two processes.

/SR 50/ Trusted Path

An integer and confidential communication mechanism between users and processes is required to prevent unauthorized processes from eavesdropping security-critical data.

/SR 60/ Trusted Hardware Access

Only trusted code must be able to access security-critical hardware components (e.g., the CPU in protected mode, or busmaster devices).

7 References

- [1] DROPS. DICE DCE IDL grammar specification. <http://os.inf.tu-dresden.de/dice/grammar-dce.xml>, 2005.
- [2] DROPS. DICE IDL compiler. <http://os.inf.tu-dresden.de/dice/>, 2005.
- [3] DROPS. DOpE. <http://os.inf.tu-dresden.de/~nf2/dope.html>, 2005.
- [4] DROPS. Fiasco microkernel. <http://os.inf.tu-dresden.de/fiasco/>, 2005.
- [5] DROPS. L4 enviroment. <http://os.inf.tu-dresden.de/l4env/>, 2005.
- [6] DROPS. L4linux. <http://os.inf.tu-dresden.de/L4/LinuxOnL4/>, 2005.
- [7] PFITZMANN, B., RIORDAN, J., STÜBLE, C., WAIDNER, M., AND WEBER, A. The PERSEUS system architecture. Tech. rep., IBM Research Division, Zurich Laboratory, 2001.