

Design and Implementation of a Device Encryption Architecture for High Security Requirements

Oskar Senft
o.senft@sirrix.com

Marcel Winandy
winandy@crypto.rub.de

Marcel Selhorst
selhorst@crypto.rub.de

Marko Wolf
mwolf@crypto.rub.de

Christian Stüble
stueble@acm.org

Michael Scheibel
m.scheibel@sirrix.com

November 2005

Abstract

This technical report presents the design and implementation of an encryption system for very high security requirements.

The proposed architecture isolates security-critical operations and encryption keys from the legacy operating system and its applications and thus remedies many deficiencies of current systems. We define security requirements as well as functional requirements and show that our solution fulfills these requirements. The proposed architecture has been developed in the context of the European Multilaterally Secure Computing Base Project (EMSCB). A prototype is available and constantly improved.

1 Introduction

More and more sensitive data is stored on consumer PCs today, including business plans, authorization secrets, and email correspondence. If the PC is stolen or lost this data may be compromised.

An approved security mechanism to mitigate this risk is to encrypt the data. Common software encryption systems keep the encryption key in kernel memory [14, 6, 7]. Unfortunately, current monolithic operating systems provide a large attack surface due to their conceptual weaknesses and their huge complexity. An attacker may readout the encryption key from kernel memory or simply deactivate the encryption system by exploiting a common security hole. Runtime protections such as access control and user authentication may be easily circumvented by booting an alternate operating system. Furthermore, an untrusted system administrator usually has full access to all system resources including the cryptographic keys of the users. Coun-

termeasures such as mandatory role-based access control (e.g. SELinux) protect this information from a "root spy" but are much too complicated to maintain and evaluate [2].

This paper proposes a solution to this problem by providing a secure, reliable and user-friendly device encryption architecture that isolates secret keys and encryption operations from the legacy operating system similar to a hardware based solution but far more cost-effective.

Moreover the encryption system uses Trusted Computing technology [1] to bind the cryptographic key to a hardware platform and to assure software integrity.

2 Deficiencies of Existing Encryption Systems

We classify existing encryption systems based on where the encryption operations takes place:

- **Application-level:** A user application encrypts single files on demand.
- **Kernel-level:** An operating system driver encrypts file system blocks transparently.
- **Hardware-level:** Both file based and block based encryption may use a separate hardware cryptographic coprocessor to isolate cryptographic operations and associated keys.

Obviously security (and cost) increases as we go down the hierarchy (cf. Figure 1). Security problems arise when secret keys are compromised and/or encryption operations are intentionally modified.

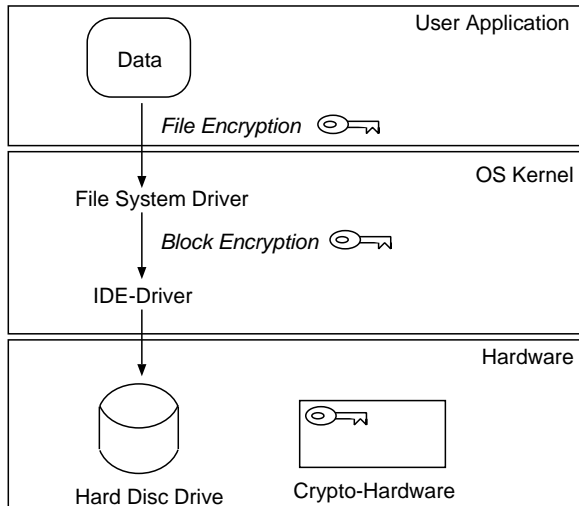


Figure 1: Classes of existing encryption systems

3 Definiton of Roles

We define two roles that interact with the encryption system:

- **Owner:** An owner can modify the configuration of the device encryption module: he can create and delete user accounts, grant and revoke access rights for devices.
- **User:** A user can register, access and deregister encrypted resources¹ 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 device encryption module.

4 Requirements

This section summarizes essential requirements for a secure device encryption system [5, 9].

4.1 Security Requirements

The main security objective of a device encryption system is to protect the confidentiality of data resulting in the following requirements:

- **User authentication:** Only authorized users should be able to access sensitive data, i.e. authorization is required at boot time and when waking up from hibernation. The authorization should withstand long-term attacks in case of system theft or loss (two-factor-authorization). Finally, an authorized user should be able to change her authorization secret or authorization token.

¹A resource identifies a physical or logical device.

- **System integrity:** The encryption system cannot be deactivated or tampered with, i.e. the integrity of the encryption system should be protected at runtime and checked at boot time. This requirement pertains to the user interface as well, e.g. the user should be able to trust the input path to the encryption system.
- **Confidentialiy of encryption keys:** The encryption key should be bound to a certain hardware platform and/or software components and isolated from the legacy operating system to make a key compromise as difficult as possible.
- **Cryptographic algorithms:** The key generation algorithm (hash computation/random number generation) should comply with current requirements . The encryption algorithm should be an open, approved standard. Its operation mode and the key length should provide reasonable security. Besides, the cryptographic algorithms can be securely updated to meet future requirements.

An additional security objective is to ensure the availability of data in case of password or token loss (encryption key recovery mechanism). If Trusted Computing technology is employed, modifications of hardware or software components - intentional or unintentional - may also lead to availability problems.

4.2 Functional Requirements

A modern device encryption architecture should as well fulfill some functional requirements such as

- **Multi-user support:** Multiple users should be able to use the encryption system on a single system. As an example, two users should be able to access a single encrypted resource despite using two different authorization secrets. A system owner should be able to grant and revoke access rights to sensitive data and administration rights to a user.
- **User friendliness:** A graphical user interface (GUI) should ease user authorization and administration.
- **User transparency:** The user should not have to decide to which data the encryption is applied. Often the user does not know which files on a system contain sensitive data (e.g. swap files, log files, backups, hibernation files, or temporary files). Mobile storage media (e.g. USB memory sticks) should be encrypted on demand.

- **Cost-effectivity:** The encryption system should not rely on cryptographic hardware extensions.
- **Performance:** The performance impact on the overall system should be minimal.
- **Reliability:** The encryption system should not accidentally corrupt the data it operates on.

5 Related Work

5.1 Commercial software device encryption products

Commercial software device encryption systems [6, 15, 16, 7] provide

- AES-128 or AES-256 encryption/decryption,
- centralized user administration and policy enforcement,
- key recovery mechanisms,
- two-factor pre-boot authentication (eToken + eToken password) (partially),
- integration of a Trusted Platform Module (TPM) to bind encryption keys to hardware and/or software components and for secure random number generation (partially),
- multi-user support for sharing resources without sharing credentials (partially).

We emphasize that none of these systems isolate the encryption keys and operations from the operating system. A representative example of a commercial hard disk encryption system is described next.

5.2 Secure Startup - Full Volume Encryption

Secure Startup - Full Volume Encryption is a hard disk encryption system integrated into the upcoming client version release of Microsoft's Windows Operating System ("Windows Vista").

The feature uses a Trusted Platform Module (TPM) 1.2 to bind the encryption key to the boot stack, thus ensuring that system files have not been tampered with while the system was offline. Secure Startup is transparent to the user as it encrypts the entire Windows volume including all user and system files. Interestingly the boot partition should be as large as 50 MB and contains a TPM-aware bootmanager which measures the integrity of all OS components before passing control to them. Secure

Startup does not use TPM authentication mechanisms but relies on conventional OS authentication after the system integrity has been verified. Secure Startup will not work properly if another application sets the authorization secret of the TPM Storage Root Key (SRK) to a value unequal to 20 bytes of zero [8].

5.3 Enforcer

The Enforcer [2, 3, 4] is a Linux Security Module (LSM) that binds the cryptographic key for an encrypted file system to long-lived system components, such as

- the Linux kernel,
- the boot stack,
- the Enforcer LSM,
- the public key of a so-called "security admin".

The security admin issues and digitally sign a list of file hashes. This security configuration is used by the Enforcer LSM to check the integrity of the applications before execution. If this check fails the encrypted file system will be unmounted.

The Enforcer even provides a mechanism to guarantee the freshness of a security configuration. To verify the integrity of the long-lived components the Enforcer enhances the LILO boot loader with TPM support.

5.4 Device Mapper with Crypt Target

The Device Mapper is a Linux 2.6 kernel feature that allows to create a virtual block device whose sectors are mapped to sectors on a physical block device. Available mapping types include mirroring, snapshotting, and encryption. Thus data written to the virtual device is transparently encrypted and passed on to the physical device (and vice versa). The crypt target (dm_crypt) uses the Linux 2.6 Cryptographic API which provides state-of-the-art symmetric ciphers and hash computation algorithms such as AES and SHA-256. The initialization vector for CBC operation mode is retrieved from the sector number.

A current Linux kernel is able to address a large number of devices as block devices, including hard disk partitions, floppy disks, regular files, USB devices, RAM disks, CR-RW, DVD-RW, DVD+RW, and network block devices. Network block devices allow transparent access to block devices exported by a server.

6 The EMSCB Project

The European Multilaterally Secure Computing Base (EMSCB) project aims at developing a trustworthy computing platform with open standards that solves many security problems of conventional platforms. The platform deploys

- hardware functionalities provided by Trusted Computing,
- a security kernel based on microkernel and
- an efficient migration of existing operating systems.

In the sense of multilateral security, the EMSCB platform allows the enforcement of security policies of different parties, i.e., end-users as well as industry. Consequently, the platform enables the realisation of various innovative business models, particularly in the area of Digital Rights Management, while averting the potential risks of Trusted Computing platforms concerning privacy issues. The sourcecode of the EMSCB platform will be published under an open-source license, e.g. the GPL.

The EMSCB project is partly funded by the German Federal Ministry of Economics and Labour. Project partners include several universities and industry organizations. This consortium is lead by Ruhr-University Bochum (Applied Data Security Group).

6.1 Basic System Model

One main design goal of EMSCB is the realization of a minimal and therefore manageable, stable and evaluable security kernel for conventional hardware platforms such as IBM-PCs, servers, embedded systems, and mobile devices like PDAs and smartphones. This requirement is fulfilled by extracting security-critical operations and data and integrate them into the security kernel. The resulting architecture is illustrated in Figure 2.

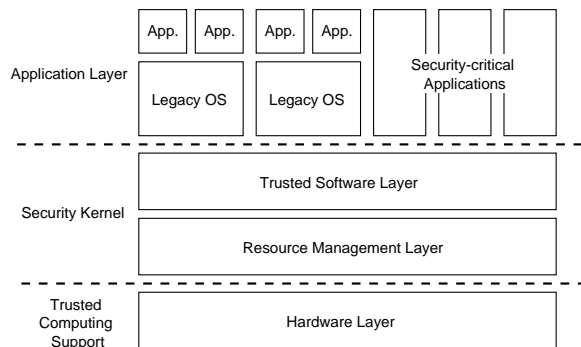


Figure 2: Overview

The EMSCB security architecture includes the following abstraction layers:

Hardware: The hardware layer includes conventional hardware like a CPU, memory, and hardware devices. Moreover, the hardware layer provides Trusted Computing technology, e.g., a TPM.

Resource Management: The main task of the resource management layer is the provision of an abstract interface of the underlying hardware resources like interrupts, memory and hardware devices. Moreover, this layer allows to share these resources and realizes access control enforcement on the object types known to this layer.

Trusted Software: By efficiently combining the services provided by the hardware layer and the resource management layer, the trusted software layer (TSL) extends the interfaces of the underlying services with security properties and ensures isolation of the applications executed on top of this layer. Examples of security services are a secure user interface (trusted GUI, trusted path), secure booting, and mutually trusted storage.

Applications: On top of the trusted software layer, security-critical and non-critical applications are executed in parallel. Legacy operating systems can be executed as isolated applications on top of the trusted software layer to provide end-users a common user interface and a backward-compatible application binary interface (ABI) and allows application providers to reuse existing non-critical applications and components.

To obtain full user transparency the encryption system has to be completely integrated into the security kernel.

7 Design

The building blocks of the proposed architecture are compartmentalization, encapsulation and simplicity to ease future security evaluations. The encryption system including its keys is strongly isolated from the operating system. Furthermore the encryption system relies on a minimum code base (the so-called Trusted Computing Base²) and is

²We define the Trusted Computing Base (TCB) as the minimum security-critical code base that needs to be trusted in enforcing a designated security policy.

highly modularized. Data flow between the modules is enabled by a central component, i.e. a microkernel. The microkernel provides only essential services such as inter-process communication (IPC), scheduling, and memory management. Device drivers run in user mode and are treated by the microkernel just like any other application. Since the applications (called "servers" in microkernel language) run in their own address space, they are strongly isolated from each other. To facilitate future security evaluations the server interfaces are as simple as possible.

As illustrated in Figure 3 the platform consists of

- Trusted Computing technology (i.e. a TPM),
- the microkernel,
- the Trusted Computing Base comprising DMA-enabled³ drivers (e.g. an IDE driver), a secure storage module, and a trusted GUI (among others),
- the trusted device encryption module,
- the legacy OS including legacy applications (e.g. Linux/Linux applications).

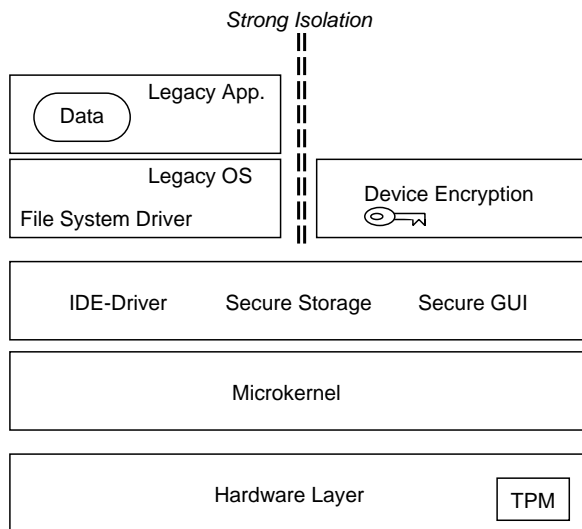


Figure 3: Design overview

The legacy operating system passes plain (encrypted) data to the encryption module which encrypts (decrypts) the data and transfers the encrypted (decrypted) data back to the legacy operating system.

³Direct Memory Access technology allows drivers to directly access arbitrary RAM regions and thus imposes a security risk on the overall system [10].

7.1 TPM integration

The secure storage module binds application data to a user authorization secret, hardware components or the trusted software modules. Binding to hardware and/or software components requires a trusted hardware component, e.g. a TPM [1].

The TPM uses on-chip registers (Platform Configuration Registers, PCRs) to store measurements (i.e. hash values) of hardware and software components securely. The TPM seal command may subsequently bind data to these PCRs. The resulting "blob" (**binary loadable object**) is then stored persistently. The data can only be revealed by a TPM unseal operation iff the PCRs specified at sealing contain the same values at unsealing AND the integrity check on the sealed data succeeds.

For our application certain PCRs should reflect the integrity of the trusted components. This can be achieved as follows (cf. Figure 4):

1. A TPM-aware (trusted) BIOS measures the MBR before execution.
2. The bootloader measures each boot stage before execution.
3. The bootloader is completely loaded. The PCRs now reflect the integrity of the boot process ("authenticated boot").
4. The trusted software components are digitally signed. The bootloader checks their signatures before execution. The corresponding public key is hard-coded into the bootloader. If a signature check fails the PCR values are invalidated and the user is requested for interaction ("secure boot").

The alternation of authenticated and secure boot allows secure updating of system components without "resealing" of secrets [2, 3, 4].

8 Implementation

The microkernel environment consists of a Fiasco microkernel which implements the L4V2 microkernel API and some supporting modules such as a graphical user interface (DOPE [19]) and an IDE driver. The family of L4-microkernels is superior to first generation microkernels (e.g. Mach) concerning performance and manageability [10].

Three operation modes have been implemented:

- **Single-user mode with pre-boot authentication:**

The user is asked to enter a password at boot time. This password is directly passed on to the encryption module before the legacy OS is booted.