Security Target for SUSE Linux Enterprise Server 15 SP2 NIAP OSPP Compliance

Version:	**0.15**
Revision:	**8**
Status:	**released**
Last Update:	**2021-08-27**
Classification:	**SUSE and atsec public**

Trademarks

Legal Notice

Revision History

Revision	Date	Author(s)	Changes to Previous Revision
0.15	2021-08-27	Stephan Mueller	Clarify FCS_CKM_EXT.4

Table of Contents

List of Tables

1 Introduction

1.1 Security Target Identification

Title:	Security Target for SUSE Linux Enterprise Server 15 SP2 NIAP OSPP Compliance
Version:	0.15
Revision:	8
Status:	released
Date:	2021-08-27
Sponsor:	SUSE, LLC
Developer:	SUSE, LLC
Certification Body:	BSI
Certification ID:	BSI-DSZ-CC-1168
Keywords:	Security Target, Common Criteria, Linux Distribution

1.2 TOE Identification

The TOE is SUSE Linux Enterprise Server Version 15 SP2.

1.3 TOE Type

The TOE type is Linux-based general-purpose operating system.

1.4 TOE Overview

1.4.1 Configurations defined with this ST

This security target documents the security characteristics of the SUSE Linux Enterprise Server distribution (abbreviated with SLES throughout this document).

1.4.2 Overview description

SUSE Linux Enterprise Server is a highly-configurable Linux-based operating system which has been developed to provide a good level of security as required in commercial environments. It also meets all requirements of the Operating System Protection Profile [OSPP].

1.4.3 Allowed Unclaimed Functionality

The TOE implements mechanisms without any security claims specified in this Security Target. This section outlines such mechanism which are allowed to be used in the evaluated configuration. As these listed mechanisms may interfere with the operation of the claimed security functionality, the evaluation ensures that the interference does not weaken any security functionality.

SLES offers full virtualization using the KVM framework. The virtual machines execute as unprivileged processes on the Linux operating system. SLES provides management interfaces to the KVM functionality.

SLES provides userspace virtualization environment called Linux Containers based on the Linux namespace and Linux control groups technology. SLES implements the host system for the Linux Containers and manages these containers.

The Linux Container technology separates the runtime environment of applications from each other. Resources accessible by applications cannot be shared with other applications. Non-shared resources include PIDs, network, IPC, user IDs, hostname, and mount points. The Linux kernel enforces the separation of these resources but provides other applications. Therefore, the Linux kernel supports the concurrent execution of Linux Containers and regular applications.

1.4.4 Required Hardware and Software

The following hardware / firmware allows the installation of the TOE:

- x86 64bit Intel Xeon processors:
  - Delta D20x-M1-PC-32-8-96GB-1TB-2x1G
- x86 64bit AMD EPYC processors:
  - AMD EPYC DP Server R181-Z90
- IBM Z System based on z/Architecture processors:
  - IBM Z System z15
- System based on ARM processors:
  - Gigabyte R181-T90

1.4.5 Intended Method of Use

1.4.5.1 General-purpose computing environment

The TOE is a Linux-based multi-user multi-tasking operating system. The TOE may provide services to several users at the same time. After successful login, the users have access to a general computing environment, allowing the start-up of user applications, issuing user commands at shell level, creating and accessing files. The TOE provides adequate mechanisms to separate the users and protect their data. Privileged commands are restricted to administrative users.

The TOE is intended to operate in a networked environment with other instantiations of the TOE as well as other well-behaved peer systems operating within the same management domain. All those systems need to be configured in accordance with a defined common security policy.

It is assumed that responsibility for the safeguarding of the user data protected by the TOE can be delegated to human users of the TOE if such users are allowed to log on and spawn processes on their behalf. All user data is under the control of the TOE. The user data is stored in named objects, and the TOE can associate a description of the access rights to that object with each named object.

The TOE enforces controls such that access to data objects can only take place in accordance with the access restrictions placed on that object by its owner, and by administrative users. Ownership of named objects may be transferred under the control of the access control policies implemented by the TOE.

Discretionary access rights (e.g. read, write, execute) can be assigned to data objects with respect to subjects identified with their UID, GID and supplemental GIDs. Once a subject is granted access to an object, the content of that object may be used freely to influence other objects accessible to this subject.

SLES has significant security extensions compared to standard UNIX systems:

- Access Control Lists
- Labels assigned to a number of kernel objects within a security context defined and managed by the AppArmor security module
- Block device encryption and ensuring the confidentiality of data at rest

1.4.5.2 Operating environment

The TOE permits one or more processors and attached peripheral and storage devices to be used by multiple applications assigned to different UIDs to perform a variety of functions requiring controlled shared access to the data stored on the system. With different UIDs proper access restrictions to resources assigned to processes can be enforced using the access control mechanisms provided by the TOE. Such installations and usage scenarios are typical for systems accessed by processes or users local to, or with otherwise protected access to, the computer system.

Note: The TOE provides the platform for installing and running arbitrary services. These additional services are not part of the TOE. The TOE is solely the operating system which provides the runtime environment for such services.

All human users, if existent, as well as all services offered by SLES are assigned unique user identifiers within the single host system that forms the TOE. This user identifier is used together with the attributes assigned to the user identifier as the basis for access control decisions. Except for virtual machine accesses, the TOE authenticates the claimed identity of the user before allowing the user to perform any further actions. Services may be spawned by the TOE without the need for user-interaction. The TOE internally maintains a set of identifiers associated with processes, which are derived from the unique user identifier upon login of the user or from the configured user identifier for a TOE-spawned service. Some of those identifiers may change during the execution of the process according to a policy implemented by the TOE.

1.4.6 Major Security Features

The primary security features of the TOE are specified as part of the logical boundary description.

These primary security features are supported by domain separation and reference mediation, which ensure that the features are always invoked and cannot be bypassed.

1.5 TOE Description

1.5.1 Introduction

SLES is a general purpose, multi-user, multi-tasking Linux based operating system. It provides a platform for a variety of applications.

The SLES evaluation covers a potentially distributed network of systems running the evaluated versions and configurations of SLES as well as other peer systems operating within the same management domain. The hardware platforms selected for the evaluation consist of machines which are available when the evaluation has completed and to remain available for a substantial period of time afterwards.

The TOE Security Functions (TSF) consist of functions of SLES that run in kernel mode plus a set of trusted processes. These are the functions that enforce the security policy as defined in this Security Target. Tools and commands executed in user mode that are used by an administrative user need also to be trusted to manage the system in a secure way. But as with other operating system evaluations they are not considered to be part of this TSF.

The hardware, the BIOS firmware and potentially other firmware layers between the hardware and the TOE are considered to be part of the TOE environment.

The TOE includes standard networking applications, including applications allowing access of the TOE via cryptographically protected communication channels, such as SSH.

System administration tools include the standard command line tools. A graphical user interface for system administration or any other operation is not included in the evaluated configuration.

The TOE environment also includes applications that are not evaluated, but are used as unprivileged tools to access public system services. For example a network server using a port above 1024 may be used as a normal application running without root privileges on top of the TOE. The additional documentation specific for the evaluated configuration provides guidance how to set up such applications on the TOE in a secure way.

1.5.2 TOE boundaries

1.5.2.1 Physical

The Target of Evaluation is based on the following system software:

- SUSE Linux Enterprise Server in the above mentioned version

The TOE and its documentation are supplied on ISO images distributed via the SUSE Portal. The TOE includes a package holding the additional user and administrator documentation.

In addition to the installation media, the following documentation is provided:

- Evaluated Configuration Guide published by SUSE at the end of the evaluation
- Manual pages for all applications, configuration files and system calls

The hardware applicable to the evaluated configuration is listed above at page 9. The analysis of the hardware capabilities as well as the firmware functionality is covered by this evaluation to the extent that the following capabilities supporting the security functionality are analyzed and tested:

- Memory separation capability
- Unavailability of privileged processor states to untrusted user code (like the hypervisor state or the SMM)
- Full testing of the security functionality on all listed hardware systems

1.5.2.2 Logical

The primary security features of the TOE are:

Auditing

The Lightweight Audit Framework (LAF) is designed to be an audit system making Linux compliant with the requirements from Common Criteria. LAF is able to intercept all system calls as well as retrieving audit log entries from privileged user space applications. The subsystem allows configuring the events to be actually audited from the set of all events that are possible to be audited.

The TOE can be deployed as an audit server that receives audit logs from other TOE instances. These audit logs are stored locally. The TOE provides search and review facilities to authorized administrators for all audit logs.

Cryptographic support

The TOE provides cryptographically secured communication to allow remote entities to log into the TOE. For interactive usage, the SSHv2 protocol is provided. The TOE provides the server side as well as the client side applications. Using OpenSSH, password-based and public-key-based authentication are allowed.

In addition, the TOE provides confidentiality protected data storage using the device mapper target dm_crypt. Using this device mapper target, the Linux operating system offers administrators and users cryptographically protected block device storage space. With the help of a Password-Based Key-Derivation Function version 2 (PBKDF2) implemented with the LUKS mechanism, a user-provided passphrase protects the volume key which is the symmetric key for encrypting and decrypting data stored on disk. Any data stored on the block devices protected by dm_crypt is encrypted and cannot be decrypted unless the volume key for the block device is decrypted with the passphrase processed by PBKDF2. With the device mapper mechanism, the TOE allows for transparent encryption and decryption of data stored on block devices, such as hard disks.

Also, the TOE offers the TLS protocol to protect network links to remote systems. The TLS protocol stack can be used as TLS client.

Identification and Authentication

User identification and authentication in the TOE includes all forms of interactive login (e.g. using the SSH protocol or log in at the local console) as well as identity changes through the su or sudo command. These all rely on explicit authentication information provided interactively by a user.

The authentication security function allows password-based authentication. For SSH access, public-key-based authentication is also supported.

Password quality enforcement mechanisms are offered by the TOE which are enforced at the time when the password is changed.

Discretionary Access Control

DAC allows owners of named objects to control the access permissions to these objects. These owners can permit or deny access for other users based on the configured permission settings. The DAC mechanism is also used to ensure that untrusted users cannot tamper with the TOE mechanisms.

In addition to the standard Unix-type permission bits for file system objects as well as IPC objects, the TOE implements POSIX access control lists. These ACLs allow the specification of the access to individual file system objects down to the granularity of a single user.

Security Management

The security management facilities provided by the TOE are usable by authorized users and/or authorized administrators to modify the configuration of TSF.

Additional Functions

The TOE provides many more functions and mechanisms. The evaluation ensures that all these additional functions do not interfere with the above mentioned security mechanisms in the evaluated configuration. The mechanisms given in the following list, however, may interfere with the security functionality of the TOE and should be allowed in the evaluated configuration. Therefore, the evaluation assesses the functionality to verify that the impact on the security functionality at most adds further restrictions as outlined below.

- Linux Container support: The TOE offers userspace virtualization support via Linux Container. That virtualization support shall be allowed to be used such that it does not interfere with the operation of the security functions. The evaluation ensures that the constraints associated with the use of Linux Containers in the evaluated configuration guide has no adverse impact on the security functionality. In addition, the libvirt daemon is allowed to run with the privileges of the root user to allow management of Linux Containers.
- Virtual machine support: The TOE offers the KVM framework allowing the execution of virtual machines. This support is allowed to be used such that it does not interfere with the operation of the security functions. The evaluation ensures that the constraints associated with the use of KVM in the evaluated configuration has no adverse impact on the security functionality. In addition, the libvirt daemon is allowed to run with the privileges of the root user to allow management of KVM.

Additional mechanisms and functions that would interfere with the operation of the security functions are disallowed in the evaluated configuration and the Evaluation Configuration Guide provides instructions to the administrator on how to disable them. Note: TOE mechanism which provide additional restrictions to the above claimed security functions are allowed in the evaluated configuration. For example, the eCryptFS cryptographic file system provided with the TOE and permitted in the evaluated configuration even though they have not been subject to this evaluation. The eCryptFS provides further restrictions on, for example, the security function of discretionary access control mechanism for file system objects and therefore cannot breach the security functionality as the discretionary access control rules of the "lower" file system are still enforced. The following table enumerates mechanisms that are provided with the TOE but which are excluded from the evaluation:

Functions	**Exclusion discussion**
eCryptFS	eCryptFS is not allowed to be used in the evaluated configuration. The encryption capability provided with this file system is therefore unavailable to any user.
Ext4 file-based encryption	Ext4 file-based encryption is not allowed to be used in the evaluated configuration. The encryption capability provided with this file system is therefore unavailable to any user.
SMACK	The mandatory access control functionality offered by the SMACK LSM is not assessed by the evaluation and disabled in the evaluated configuration.
SELinux	The mandatory access control functionality offered by the SELinux LSM is not assessed by the evaluation and disabled in the evaluated configuration.
AppArmor	The mandatory access control functionality offered by the AppArmor LSM is not assessed by the evaluation but is allowed the evaluated configuration to support the KVM framework.
GSS-API Security Mechanisms	The GSS-API is used to secure the connection between different audit daemons. The security mechanisms used by the GSS-API, however, is not part of the evaluation. Therefore, A.CONNECT applies to the audit-related communication link.

Table 1: Non-evaluated functionalities

Note: Packages and mechanisms not covered with security claims and subsequent assessments during the evaluation or disabling the respective functionality in the evaluated configuration result from the exact conformance requirement stipulated by the chosen protection profile but does not imply that the respective package or functionality is implemented insecurely.

1.5.2.3 Configurations

The evaluated configurations are defined as follows:

- The CC evaluated package set must be selected at install time in accordance with the description provided in the Evaluated Configuration Guide and installed accordingly.
- The TOE supports the use of IPv4 and IPv6, both are also supported in the evaluated configuration. IPv6 conforms to the following RFCs:
    - RFC 2460 specifying the basic IPv6 protocol
    - IPv6 source address selection as documented in RFC 3484

- ○ Linux implements several new socket options (IPV6_RECVPKTINFO, IPV6_PKTINFO, IPV6_RECVHOPOPTS, IPV6_HOPOPTS, IPV6_RECVDSTOPTS, IPV6_DSTOPTS, IPV6_RTHDRDSTOPTS, IPV6_RECVRTHDR, IPV6_RTHDR, IPV6_RECVHOPOPTS, IPV6_HOPOPTS, IPV6_{RECV,}TCLASS) and ancillary data in order to support advanced IPv6 applications including ping, traceroute, routing daemons and others. The following section introduces Internet Protocol Version 6 (IPv6). For additional information about referenced socket options and advanced IPv6 applications, see RFC 3542
- ○ Transition from IPv4 to IPv6: dual stack, and configured tunneling according to RFC 4213.
- ○ Additional RFCs covering various cryptographic aspects are outlined as part of the Security Functional Requirements.

- The default configuration for identification and authentication are the defined password-based PAM modules as well as key based authentication for OpenSSH. Support for other authentication options, e.g. smart card authentication, is not included in the evaluation configuration.
- If the system console is used, it must be connected directly to the TOE and afforded the same physical protection as the TOE.

Deviations from the configurations and settings specified with the Evaluated Configuration Guide are not permitted.

The TOE comprises a single system (and optional peripherals) running the TOE software listed. Cluster configurations are not permitted in the evaluated configuration.

1.5.2.4 TOE Environment

Several TOE systems may be interlinked in a network, and individual networks may be joined by bridges and/or routers, or by TOE systems which act as routers and/or gateways. Each of the TOE systems implements its own security policy. The TOE does not include any synchronization function for those policies. As a result a single user may have user accounts on each of those systems with different UIDs, different roles, and other different attributes. (A synchronization method may optionally be used, but it not part of the TOE and must not use methods that conflict with the TOE requirements.)

If other systems are connected to a network they need to be configured and managed by the same authority using an appropriate security policy that does not conflict with the security policy of the TOE. All links between this network and untrusted networks (e. g. the Internet) need to be protected by appropriate measures such as carefully configured firewall systems that prohibit attacks from the untrusted networks. Those protections are part of the TOE environment.

1.5.2.5 Security Policy Model

The security policy for the TOE is defined by the security functional requirements in chapter 6. The following is a list of the subjects and objects participating in the policy.

Subjects:

- Processes acting on behalf of a human user or technical entity.

Named objects:

- File system objects in the following allowed file systems:
  - ○ BTRFS - standard file system for general data
  - ○ Ext3 - standard file system for general data
  - ○ Ext4 - standard file system for general data
  - ○ XFS - standard file system for general data

    - VFAT - special purpose file system for UEFI BIOS support mounted at /boot/efi
    - iso9660 - ISO9660 file system for CD-ROM and DVD
    - tmpfs - the temporary file system backed by RAM
    - rootfs - the virtual root file system used temporarily during system boot
    - procfs - process file system holding information about processes, general statistical data and tunable kernel parameters
    - sysfs - system-related file system covering general information about resources maintained by the kernel including several tunable parameters for these resources
    - devpts - pseudoterminal file system for allocating virtual TTYs on demand
    - devtmpfs - temporary file system that allows the kernel to generate character or block device nodes
    - binfmt_misc - configuration interface allowing the assignment of executable file formats with user space applications
    - securityfs - interface for loadable security modules (LSM) to provide tunables and configuration interfaces to user space
    - cgroup - interface for configuring the control groups mechanism provided by the kernel
    - debugfs - interface for accessing low-level kernel data

  Please note that the TOE supports a number of additional virtual (i.e. without backing of persistent storage) file systems which are only accessible to the TSF - they are not or cannot be mounted. All above mentioned virtual file systems implement access decisions based DAC attributes inferred from the underlying process’ DAC attributes. Additional restrictions may apply for specific objects in this file system.
- Inter Process Communication (IPC) objects:
    - Semaphores
    - Shared memory
    - Message queues
    - Named pipes
    - UNIX domain socket special files
- Network sockets (irrespectively of their type - such as Internet sockets and netlink sockets)
- Storage device objects (covered by dm_crypt - note that such storage device objects may be provided by either block devices or LVM devices)
- at and cron job queues maintained for each user

TSF data:

- TSF executable code
- Subject meta data - all data used for subjects except data which is not interpreted by the TSF and does not implement parts of the TSF (this data is called user data)
- Named object meta data - all data used for the respective objects except data which is not interpreted by the TSF and does not implement parts of the TSF (this data is called user data)
- User accounts
- Audit records
- Volume keys for dm_crypt block devices and passphrases protecting the session keys

User data:

- Non-TSF executable code used to drive the behavior of subjects

- Data not interpreted by TSF and stored or transmitted using named objects

1.5.3 Technical decisions

The following table enumerates the applied technical decisions raised by NIAP for the applied protection profiles. In addition, the table outlines how the technical decisions are applied to the ST.

TD	TD Summary	Application in ST
TD0578	SHA-1 is no longer mandatory	Changes applied
TD0525	Updates to Certificate Revocation (FIA_X509_EXT.1)	Changes applied
TD0501	Cryptographic selections and updates for OS PP	Changes applied
TD0496	GPOS PP adds allow-with statement for VPN Client V2.1	Change not applied as VPN functionality is not claimed
TD0493	X.509v3 certificates when using digital signatures for Boot Integrity	Change applied
TD0463	Clarification for FPT_TUD_EXT	Change applied
TD0446	Missing selections for SSH	Change applied
TD0441	Updated TLS Ciphersuites for OS PP	Change applied
TD0420	Conflict in FCS_SSHC_EXT.1.1 and FCS_SSHS_EXT.1.1	Change affects assurance activity only
TD0386	Platform-Provided Verification of Update	Change applied
TD0365	FCS_CKM_EXT.4 selections	Change applied
TD0332	Support for RSA SHA2 host keys	Change applied
TD0331	SSH Rekey Testing	Change affects assurance activity only
TD0240	SSH Rekey Testing	Change applied

2 CC Conformance Claim

This Security Target is CC Part 2 extended and CC Part 3 extended.

This Security Target claims conformance to the following Protection Profiles and PP packages:

- [OSPP]: Protection Profile for General Purpose Operating Systems. Version 4.2.1 as of 2019-04-22; exact conformance.
- [SSHEP]: Extended Package for Secure Shell (SSH). Version 1.0 as of 2016-02-19; exact conformance.

Common Criteria [CC] version 3.1 revision 5 is the basis for this conformance claim.

3 Security Problem Definition

3.1 Threat Environment

Threats to be countered by the TOE are characterized by the combination of an asset being subject to a threat, a threat agent and an adverse action.

The definition of threat agents and protected assets that follows is applicable to the OSPP unless noted otherwise.

3.1.1 Assets

Assets to be protected are:

- Persistent storage objects used to store user data and/or TSF data, where this data needs to be protected from any of the following operations:
    - Unauthorized read access
    - Unauthorized modification
    - Unauthorized deletion of the object
    - Unauthorized creation of new objects
    - Unauthorized management of object attributes
- Transient storage objects, including network data
- TSF functions and associated TSF data
- The resources managed by the TSF that are used to store the above-mentioned objects, including the metadata needed to manage these objects.

3.1.2 Threat Agents

Threat agents are external entities that potentially may attack the TOE. They satisfy one or more of the following criteria:

- External entities not authorized to access assets may attempt to access them either by masquerading as an authorized entity or by attempting to use TSF services without proper authorization.
- External entities authorized to access certain assets may attempt to access other assets they are not authorized to either by misusing services they are allowed to use or by masquerading as a different external entity.
- Untrusted subjects may attempt to access assets they are not authorized to either by misusing services they are allowed to use or by masquerading as a different subject.

Threat agents are typically characterized by a number of factors, such as expertise, available resources, and motivation, with motivation being linked directly to the value of the assets at stake. The TOE protects against intentional and unintentional breach of TOE security by attackers possessing an basic attack potential.

3.1.3 Threats countered by the TOE

T.NETWORK_ATTACK

An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may engage in communications with applications and services running on or part of the OS with the intent of compromise. Engagement may consist of altering existing legitimate communications.

T.NETWORK_EAVESDROP

An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may monitor and gain access to data exchanged between applications and services that are running on or part of the OS.

T.LOCAL_ATTACK

An attacker may compromise applications running on the OS. The compromised application may provide maliciously formatted input to the OS through a variety of channels including unprivileged system calls and messaging via the file system.

T.LIMITED_PHYSICAL_ACCESS

An attacker may attempt to access data on the OS while having a limited amount of time with the physical device.

3.2 Assumptions

3.2.1 Environment of use of the TOE

3.2.1.1 Procedural

A.PLATFORM

The OS relies upon a trustworthy computing platform for its execution. This underlying platform is out of scope of this PP.

3.2.1.2 Personnel

A.PROPER_USER

The user of the OS is not willfully negligent or hostile, and uses the software in compliance with the applied enterprise security policy. At the same time, malicious software could act as the user, so requirements which confine malicious subjects are still in scope.

A.PROPER_ADMIN

The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise security policy.

4 Security Objectives

4.1 Objectives for the TOE

O.ACCOUNTABILITY

Conformant OSes ensure that information exists that allows administrators to discover unintentional issues with the configuration and operation of the operating system and discover its cause. Gathering event information and immediately transmitting it to another system can also enable incident response in the event of system compromise.

O.INTEGRITY

Conformant OSes ensure the integrity of their update packages. OSes are seldom if ever shipped without errors, and the ability to deploy patches and updates with integrity is critical to enterprise network security. Conformant OSes provide execution environment-based mitigations that increase the cost to attackers by adding complexity to the task of compromising systems.

O.MANAGEMENT

To facilitate management by users and the enterprise, conformant OSes provide consistent and supported interfaces for their security-relevant configuration and maintenance. This includes the deployment of applications and application updates through the use of platform-supported deployment mechanisms and formats, as well as providing mechanisms for configuration and application execution control.

O.PROTECTED_STORAGE

To address the issue of loss of confidentiality of credentials in the event of loss of physical control of the storage medium, conformant OSes provide data-at-rest protection for credentials. Conformant OSes also provide access controls which allow users to keep their files private from other users of the same system.

O.PROTECTED_COMMS

To address both passive (eavesdropping) and active (packet modification) network attack threats, conformant OSes provide mechanisms to create trusted channels for CSP and sensitive data. Both CSP and sensitive data should not be exposed outside of the platform.

4.2 Objectives for the Operational Environment

OE.PLATFORM

The OS relies on being installed on trusted hardware.

OE.PROPER_USER

The user of the OS is not willfully negligent or hostile, and uses the software within compliance of the applied enterprise security policy. Standard user accounts are provisioned in accordance with the least privilege model. Users requiring higher levels of access should have a separate account dedicated for that use.

OE.PROPER_ADMIN

The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise security policy.

4.3 Security Objectives Rationale

4.3.1 Coverage

The following table provides a mapping of TOE objectives to threats and policies, showing that each objective counters or enforces at least one threat or policy, respectively.

Objective	Threats / OSPs
O.ACCOUNTABILITY	T.NETWORK_ATTACK T.LOCAL_ATTACK
O.INTEGRITY	T.NETWORK_ATTACK T.LOCAL_ATTACK
O.MANAGEMENT	T.NETWORK_ATTACK T.NETWORK_EAVESDROP
O.PROTECTED_STORAGE	T.LIMITED_PHYSICAL_ACCESS
O.PROTECTED_COMMS	T.NETWORK_ATTACK T.NETWORK_EAVESDROP

Table 2: Mapping of security objectives to threats and policies

The following table provides a mapping of the objectives for the Operational Environment to assumptions, threats and policies, showing that each objective holds, counters or enforces at least one assumption, threat or policy, respectively.

Objective	Assumptions / Threats / OSPs
OE.PLATFORM	A.PLATFORM
OE.PROPER_USER	A.PROPER_USER
OE.PROPER_ADMIN	A.PROPER_ADMIN

Table 3: Mapping of security objectives for the Operational Environment to assumptions, threats and policies

4.3.2 Sufficiency

The following rationale provides justification that the security objectives are suitable to counter each individual threat and that each security objective tracing back to a threat, when achieved, actually contributes to the removal, diminishing or mitigation of that threat.

Threat	Rationale for security objectives
T.NETWORK_ATTACK	The threat T.NETWORK_ATTACK is countered by O.PROTECTED_COMMS as this provides for integrity of transmitted data. The threat T.NETWORK_ATTACK is countered by O.INTEGRITY as this provides for integrity of software that is installed onto the system from the network. The threat T.NETWORK_ATTACK is countered by O.MANAGEMENT as this provides for the ability to configure the OS to defend against network attack. The threat T.NETWORK_ATTACK is countered by O.ACCOUNTABILITY as this provides a mechanism for the OS to report behavior that may indicate a network attack has occurred.

Threat	Rationale for security objectives
T.NETWORK_EAVESDROP	The threat T.NETWORK_EAVESDROP is countered by O.PROTECTED_COMMS as this provides for confidentiality of transmitted data. The threat T.NETWORK_EAVESDROP is countered by O.MANAGEMENT as this provides for the ability to configure the OS to protect the confidentiality of its transmitted data.
T.LOCAL_ATTACK	The objective O.INTEGRITY protects against the use of mechanisms that weaken the TOE with regard to attack by other software on the platform. The objective O.ACCOUNTABILITY protects against local attacks by providing a mechanism to report behavior that may indicate a local attack is occurring or has occurred.
T.LIMITED_PHYSICAL_ACCESS	The objective O.PROTECTED_STORAGE protects against unauthorized attempts to access physical storage used by the TOE.

Table 4: Sufficiency of objectives countering threats

The following rationale provides justification that the security objectives for the environment are suitable to cover each individual assumption, that each security objective for the environment that traces back to an assumption about the environment of use of the TOE, when achieved, actually contributes to the environment achieving consistency with the assumption, and that if all security objectives for the environment that trace back to an assumption are achieved, the intended usage is supported.

Assumption	Rationale for security objectives
A.PLATFORM	The operational environment objective OE.PLATFORM is realized through A.PLATFORM.
A.PROPER_USER	The operational environment objective OE.PROPER_USER is realized through A.PROPER_USER.
A.PROPER_ADMIN	The operational environment objective OE.PROPER_ADMIN is realized through A.PROPER_ADMIN.

Table 5: Sufficiency of objectives holding assumptions

5 Extended Components Definition

The Security Target uses the extended components defined by [OSPP] and [SSHEP]. They are not re-defined here again.

6 Security Requirements

6.1 TOE Security Functional Requirements

All of the following SFRs are derived from the OSPP.

The operations of assignments and selections are marked with bold font. The operation of refinement is marked with strike through (deletion) or italics (addition). Iterations are marked with an ID added to the SFR number.

The table below summarizes the SFRs for the TOE and the operations performed on the components according to CC part 1. Operations in the SFRs use the following convention:

- Iterations (Iter.) are identified by appending a suffix to the original SFR.
- Refinements (Ref.) added to the text are shown in *italic text*, deletions are shown as ~~strikethrough text~~.
- Assignments (Ass.) are shown in **bold text**.
- Selections (Sel.) are shown in **bold text**.

Security functional group	Security functional requirement	Base security functional component	Source	Operations			
				Iter.	Ref.	Ass.	Sel.
General-purpose computing environment	FCS_CKM.1 Cryptographic key generation		OSPP	No	No	No	Yes
	FCS_CKM.2 Cryptographic key establishment		OSPP	No	No	No	Yes
	FCS_CKM_EXT.4 Cryptographic key destruction		OSPP	No	No	No	Yes
	FCS_COP.1(1) Cryptographic Operation - Encryption/Decryption	FCS_COP.1	OSPP	Yes	No	No	Yes
	FCS_COP.1(2) Cryptographic Operation - Hashing	FCS_COP.1	OSPP	Yes	No	No	Yes
	FCS_COP.1(3) Cryptographic Operation - Signing	FCS_COP.1	OSPP	Yes	No	No	Yes
	FCS_COP.1(4) Cryptographic Operation - Keyed-Hash Message Authentication	FCS_COP.1	OSPP	Yes	No	Yes	Yes
	FCS_RBG_EXT.1 Random Bit Generation		OSPP	No	No	No	Yes
	FCS_STO_EXT.1 Storage of Sensitive Data		OSPP	No	No	No	No
	FCS_TLSC_EXT.1 TLS Client Protocol		OSPP	No	No	No	Yes
	FCS_TLSC_EXT.2 TLS Client Protocol	FCS_TLSC_EXT.1	OSPP	No	No	No	Yes
	FDP_ACF_EXT.1 Access Controls for Protecting User Data		OSPP	No	No	No	No

Security functional group	Security functional requirement	Base security functional component	Source	Operations			
				Iter.	Ref.	Ass.	Sel.
	FMT_MOF_EXT.1 Management of security functions behavior		OSPP	No	No	No	No
	FMT_SMF_EXT.1 Specification of Management Functions		OSPP	No	No	Yes	Yes
	FPT_ACF_EXT.1 Access controls		OSPP	No	No	Yes	No
	FPT_ASLR_EXT.1 Address Space Layout Randomization		OSPP	No	No	Yes	Yes
	FPT_SBOP_EXT.1 Stack Buffer Overflow Protection		OSPP	No	No	No	Yes
	FPT_TST_EXT.1 Boot Integrity		OSPP	No	No	No	Yes
	FPT_TUD_EXT.1 Trusted Update		OSPP	No	No	No	Yes
	FPT_TUD_EXT.2 Trusted Update for Application Software		OSPP	No	No	No	No
	FAU_GEN.1 Audit data generation		OSPP	No	No	Yes	Yes
	FIA_AFL.1 Authentication failure handling		OSPP	No	No	Yes	Yes
	FIA_UAU.5 Multiple Authentication Mechanisms		OSPP	No	No	Yes	Yes
	FIA_X509_EXT.1 X.509 Certificate Validation		OSPP	No	No	No	Yes
	FIA_X509_EXT.2 X.509 Certificate Authentication		OSPP	No	No	No	Yes
	FTP_ITC_EXT.1 Trusted channel communication		OSPP	No	No	No	Yes
	FTP_TRP.1 Trusted Path		OSPP	No	No	No	Yes
Extended Package for Secure Shell	FCS_COP.1(5) Cryptographic operation - Encryption / Decryption	FCS_COP.1	SSHEP	Yes	No	No	Yes
	FCS_SSH_EXT.1 SSH Protocol		SSHEP	No	No	No	Yes
	FCS_SSHC_EXT.1 SSH Protocol - Client		SSHEP	No	No	Yes	Yes
	FCS_SSHS_EXT.1 SSH Protocol - Server		SSHEP	No	No	Yes	Yes

Table 6: SFRs for the TOE

6.1.1 General-purpose computing environment

6.1.1.1 Cryptographic key generation (FCS_CKM.1)

FCS_CKM.1.1 The OS shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm

1. **RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.3,**
2. **ECC schemes using "NIST curves" P-256, P-384 and P-521 that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.4,**
3. **FFC schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.1.**
4. **FFC Schemes using safe primes that meet the following: ‘NIST Special Publication 800-56A Revision 3, “Recommendation for Pair-Wise Key Establishment Schemes**

6.1.1.2 Cryptographic key establishment (FCS_CKM.2)

FCS_CKM.2.1 The OS shall implement functionality to perform cryptographic key establishment in accordance with a specified cryptographic key establishment method:

1. **RSA-based key establishment schemes that meets the following: RSAES-PKCS1-v1_5 as specified in Section 7.2 of RFC 8017, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2,**
2. **Elliptic curve-based key establishment schemes that meets the following: NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography",**
3. **Finite field-based key establishment schemes that meets the following: NIST Special Publication 800-56A Revision 3, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”.**

6.1.1.3 Cryptographic key destruction (FCS_CKM_EXT.4)

FCS_CKM_EXT.4.1 The OS shall destroy cryptographic keys and key material in accordance with a specified cryptographic key destruction method

1. **For volatile memory, the destruction shall be executed by a single overwrite consisting of zeroes,**
2. **For non-volatile memory that consists of the invocation of an interface provided by the underlying platform that instructs the underlying platform to destroy the abstraction that represents the key.**

FCS_CKM_EXT.4.2 The OS shall destroy all keys and key material when no longer needed.

6.1.1.4 Cryptographic Operation - Encryption/Decryption (FCS_COP.1(1))

FCS_COP.1.1 The OS shall perform [encryption/decryption services for data] in accordance with a specified cryptographic algorithm

1. **AES-XTS (as defined in NIST SP 800-38E),**
2. **AES-CBC (as defined in NIST SP 800-38A)**

and

1. **AES-GCM (as defined in NIST SP 800-38D)**

and cryptographic key sizes

1. **128-bit**
2. **256-bit.**

6.1.1.5 Cryptographic Operation - Hashing (FCS_COP.1(2))

FCS_COP.1.1 The OS shall perform [cryptographic hashing services] in accordance with a specified cryptographic algorithm

1. **SHA-1,**
2. **SHA-256,**
3. **SHA-384,**
4. **SHA-512,**

and message digest sizes 160 bits and

1. **256 bits,**
2. **384 bits,**
3. **512 bits**

that meet the following: [FIPS Pub 180-4].

6.1.1.6 Cryptographic Operation - Signing (FCS_COP.1(3))

FCS_COP.1.1 The OS shall perform [cryptographic signature services (generation and verification)] in accordance with a specified cryptographic algorithm

1. **RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Section 4,**
2. **ECDSA schemes using "NIST curves" P-256, P-384 and P-521 that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Section 5.**

6.1.1.7 Cryptographic Operation - Keyed-Hash Message Authentication (FCS_COP.1(4))

FCS_COP.1.1 The OS shall perform [keyed-hash message authentication services] in accordance with a specified cryptographic algorithm

1. **SHA-1,**
2. **SHA-256,**
3. **SHA-384,**
4. **SHA-512**

with key sizes **equal to the message digest size of the chosen cipher** and message digest sizes

1. **160 bits,**

2. **256 bits,**
3. **384 bits,**
4. **512 bits**

that meet the following: [FIPS Pub 198-1 The Keyed-Hash Message Authentication Code and FIPS Pub 180-4 Secure Hash Standard].

6.1.1.8 Random Bit Generation (FCS_RBG_EXT.1)

FCS_RBG_EXT.1.1 The OS shall perform all deterministic random bit generation (DRBG) services in accordance with NIST Special Publication 800-90A using

1. **CTR_DRBG (AES),**
2. **HMAC_DRBG (any).**

.

FCS_RBG_EXT.1.2 The deterministic RBG used by the OS shall be seeded by an entropy source that accumulates entropy from a

1. **software-based noise source**

with a minimum of

1. **256 bits**

of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.

Application Note CTR DRBG:

For the German Schema, the SFR is "translated" into an AIS 20/31 compliant SFR following the FCS_RNG.1 definition. Therefore, this application note states the SFR as part of this application note:

FCS_RNG.1.1: The TSF shall provide a deterministic random number generator conforming to SP800-90A CTR_DRBG with AES-256 core using a derivation function without prediction resistance that implements:

- *a) DRG2.1: If initialized with a random seed using high-resolution time stamps of block device access events, human interface device events and interrupt events as seed source, the internal state of the RNG shall have a minentropy of 256 bits.*
- *b) DRG2.2: The DRNG provides forward secrecy.*
- *c) DRG2.3: The DRNG provides backward secrecy.*

The TSF shall provide random numbers that meet:

- *a) DRG.2.4: The RNG initialized with a random seed that is equal in size as the generated random number, every time a random number is obtained generates output for which 2**19 strings of bit length 128 are mutually different with probability of greater than 1-2**-10.*
- *b) DRG.2.5: Statistical test suites cannot practically distinguish the random numbers from output sequences of an ideal RNG. The random numbers must pass test procedure A.*

Application Note HMAC DRBG:

For the German Schema, the SFR is "translated" into an AIS 20/31 compliant SFR following the FCS_RNG.1 definition. Therefore, this application note states the SFR as part of this application note:

FCS_RNG.1.1: The TSF shall provide a deterministic random number generator conforming to SP800-90A HMAC_DRBG with SHA-256 core using a derivation function without prediction resistance that implements:

a) *DRG2.1: If initialized with a random seed using high-resolution time stamps of block device access events, human interface device events and interrupt events as seed source, the internal state of the RNG shall have a minentropy of 256 bits.*

b) *DRG2.2: The DRNG provides forward secrecy.*

c) *DRG2.3: The DRNG provides backward secrecy.*

The TSF shall provide random numbers that meet:

a) *DRG.2.4: The RNG initialized with a random seed that is equal in size as the generated random number, every time a random number is obtained generates output for which 2**19 strings of bit length 128 are mutually different with probability of greater than 1-2**-10.*

b) *DRG.2.5: Statistical test suites cannot practically distinguish the random numbers from output sequences of an ideal RNG. The random numbers must pass test procedure A.*

6.1.1.9 Storage of Sensitive Data (FCS_STO_EXT.1)

FCS_STO_EXT.1.1 The OS shall implement functionality to encrypt sensitive data stored in non-volatile storage and provide interfaces to applications to invoke this functionality.

6.1.1.10 TLS Client Protocol (FCS_TLSC_EXT.1)

FCS_TLSC_EXT.1.1 The OS shall implement TLS 1.2 (RFC 5246) supporting the following cipher suites:

1. **TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246,**
2. **TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246,**
3. **TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246,**
4. **TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246,**
5. **TLS_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5288,**
6. **TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288,**
7. **TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246,**
8. **TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5288,**
9. **TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288,**
10. **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,**
11. **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,**
12. **TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289,**
13. **TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289**

.

FCS_TLSC_EXT.1.2 The OS shall verify that the presented identifier matches the reference identifier according to RFC 6125.

FCS_TLSC_EXT.1.3 The OS shall only establish a trusted channel if the peer certificate is valid.

6.1.1.11 TLS Client Protocol (FCS_TLSC_EXT.2)

FCS_STO_EXT.1.1 The OS shall present the Supported Groups Extension in the Client Hello with the following supported groups:

1. **secp256r1,**
2. **secp384r1,**
3. **secp521r1,**

6.1.1.12 Access Controls for Protecting User Data (FDP_ACF_EXT.1)

FDP_ACF_EXT.1.1 The OS shall implement access controls which can prohibit unprivileged users from accessing files and directories owned by other users.

6.1.1.13 Management of security functions behavior (FMT_MOF_EXT.1)

FMT_MOF_EXT.1.1 The OS shall restrict the ability to perform the function indicated in the "Administrator" column in FMT_SMF_EXT.1.1 to the administrator.

6.1.1.14 Specification of Management Functions (FMT_SMF_EXT.1)

FMT_SMF_EXT.1.1 The OS shall be capable of performing the following management functions:

Number	Management Function	Administrator	User
1	Enable/disable **screen lock**	X	X
2	Configure **screen lock** inactivity timeout	X	X
3	Configure local audit storage capacity	X	N
4	Configure minimum password length	X	N
5	Configure minimum number of special characters in password	X	N
6	Configure minimum number of numeric characters in password	X	N
7	Configure minimum number of uppercase characters in password	X	N
8	Configure minimum number of lowercase characters in password	X	N
9	Configure lockout policy for unsuccessful authentication attempts through **timeouts between attempts**	X	N
10	Configure host-based firewall	X	N
11	Configure name/address of directory server with which to bind	N	N
12	Configure name/address of remote management server from which to receive management settings	N	N

Number	Management Function	Administrator	User
13	Configure name/address of audit/logging server to which to send audit/logging records	X	N
14	Configure audit rules	X	N
15	Configure name/address of network time server	X	N
16	Enable/disable automatic software update	N	N
17	Configure WiFi interface	X	N
18	Enable/disable Bluetooth interface	N	N
19	Enable/disable **network interface cards**	X	N
20	**no other function**	N	N

Table 7: Management Functions

6.1.1.15 Access controls (FPT_ACF_EXT.1)

FPT_ACF_EXT.1.1 The OS shall implement access controls which prohibit unprivileged users from modifying:

a) Kernel and its drivers/modules
b) Security audit logs
c) Shared libraries
d) System executables
e) System configuration files
f) **no other objects**

FPT_ACF_EXT.1.2 The OS shall implement access controls which prohibit unprivileged users from reading:

a) Security audit logs
b) System-wide credential repositories
c) **no other objects**

6.1.1.16 Address Space Layout Randomization (FPT_ASLR_EXT.1)

FPT_ASLR_EXT.1.1 The OS shall always randomize process address space memory locations with **11 bits for stack memory addresses, 28 bits for virtual memory addresses** bits of entropy for **all ELF binaries except the kernel.**

6.1.1.17 Stack Buffer Overflow Protection (FPT_SBOP_EXT.1)

FPT_SBOP_EXT.1.1 The OS shall **employ stack-based buffer overflow protections.**

6.1.1.18 Boot Integrity (FPT_TST_EXT.1)

FPT_TST_EXT.1.1 The OS shall verify the integrity of the bootchain up through the OS kernel and **no other component** prior to its execution through the use of **a digital signature using an X509 certificate with hardware-based protection.**

6.1.1.19 Trusted Update (FPT_TUD_EXT.1)

FPT_TUD_EXT.1.1 The OS shall provide the ability to check for updates to the OS software itself and shall use a digital signature scheme specified in FCS_COP.1(3) to validate the authenticity of the response.

FPT_TUD_EXT.1.2 The OS shall **cryptographically verify** updates to itself using a digital signature prior to installation using schemes specified in FCS_COP.1(3).

6.1.1.20 Trusted Update for Application Software (FPT_TUD_EXT.2)

FPT_TUD_EXT.2.1 The OS shall provide the ability to check for updates to application software and shall use a digital signature scheme specified in FCS_COP.1(3) to validate the authenticity of the response.

FPT_TUD_EXT.2.2 The OS shall cryptographically verify the integrity of updates to applications using a digital signature specified by FCS_COP.1(3) prior to installation.

6.1.1.21 Audit data generation (FAU_GEN.1)

FAU_GEN.1.1 The OS shall be able to generate an audit record of the following auditable events:

a) Start-up and shutdown of the audit functions;

b) All auditable events for the not-specified level of audit; and

c) 1. Authentication events (Success/Failure);
   2. Use of privileged/special rights events (Successful and unsuccessful security, audit, and configuration changes);
   3. Privilege or role escalation events (Success/Failure);
   4. **a) File and object events (Successful and unsuccessful attempts to create, access, delete, modify, modify permissions)**
      b) User and Group management events (Successful and unsuccessful add, delete, modify, disable, enable, and credential change)
      c) Attempted application invocation with arguments (Success/Failure e.g. due to software restriction policy)
      d) System reboot, restart, and shutdown events (Success/Failure)
      e) Kernel module loading and unloading events (Success/Failure
      f) Administrator or root-level access events (Success/Failure)

FAU_GEN.1.2 The OS shall record within each audit record at least the following information:

a) Date and time of the event, type of event, subject identity (if applicable), and outcome (success or failure) of the event; and

b) For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST;

i. **no additional information** .

6.1.1.22 Authentication failure handling (FIA_AFL.1)

FIA_AFL.1.1 The OS shall detect when **an administrator configurable positive integer within a positive integer value** unsuccessful authentication attempts occur related to events with **authentication based on user name and password.**

FIA_AFL.1.2 When the defined number of unsuccessful authentication attempts for an account has been met, the OS shall: **Account Disablement.**

6.1.1.23 Multiple Authentication Mechanisms (FIA_UAU.5)

FIA_UAU.5.1 The OS shall provide the following authentication mechanisms **authentication based on user name and password, for use in SSH only, SSH public key-based authentication as specified by the EP for Secure Shell** to support user authentication.

FIA_UAU.5.2 The OS shall authenticate any user's claimed identity according to the **locally stored authentication credentials.**

6.1.1.24 X.509 Certificate Validation (FIA_X509_EXT.1)

FIA_X509_EXT.1.1 The OS shall implement functionality to validate certificates in accordance with the following rules:

a) RFC 5280 certificate validation and certificate path validation.

b) The certificate path must terminate with a trusted CA certificate.

c) The OS shall validate a certificate path by ensuring the presence of the basicConstraints extension and that the CA flag is set to TRUE for all CA certificates, and that any path constraints are met.

d) The TSF shall validate that any CA certificate includes caSigning purpose in the key usage field

e) The OS shall validate the revocation status of the certificate using **a Certificate Revocation List (CRL) as specified in RFC 5759**

f) The OS shall validate the extendedKeyUsage field according to the following rules:

1. Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
2. Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
3. Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the extendedKeyUsage field.
4. S/MIME certificates presented for email encryption and signature shall have the Email Protection purpose (id-kp 4 with OID 1.3.6.1.5.5.7.3.4) in the extendedKeyUsage field.
5. OCSP certificates presented for OCSP responses shall have the OCSP Signing purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the extendedKeyUsage field.

6. Server certificates presented for EST shall have the CMC Registration Authority (RA) purpose (id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28) in the extendedKeyUsage field (conditional).

FIA_X509_EXT.1.2 The OS shall only treat a certificate as a CA certificate if the basicConstraints extension is present and the CA flag is set to TRUE.

6.1.1.25 X.509 Certificate Authentication (FIA_X509_EXT.2)

FIA_X509_EXT.2.1 The OS shall use X.509v3 certificates as defined by RFC 5280 to support authentication for TLS and **no other protocols** connections.

6.1.1.26 Trusted channel communication (FTP_ITC_EXT.1)

FTP_ITC_EXT.1.1 The OS shall use **TLS as conforming to FCS_TLSC_EXT.1, SSH as conforming to the EP for Secure Shell** to provide a trusted communication channel between itself and authorized IT entities supporting the following capabilities: **audit server, management server** that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from disclosure and detection of modification of the channel data.

6.1.1.27 Trusted Path (FTP_TRP.1)

FTP_TRP.1.1 The OS shall provide a communication path between itself and **remote** users that is logically distinct from other communication paths and provides assured identification of its endpoints and protection of the communicated data from [modification, disclosure].

FTP_TRP.1.2 The OS shall permit **the TSF, local users, remote users** to initiate communication via the trusted path.

FTP_TRP.1.3 The OS shall require use of the trusted path for [[all remote administrative actions]].

6.1.2 Extended Package for Secure Shell

6.1.2.1 Cryptographic operation - Encryption / Decryption (FCS_COP.1(5))

FCS_COP.1.1 The SSH software shall **perform** encryption/decryption services for data in accordance with a specified cryptographic algorithm AES-CTR (as defined in NIST SP 800-38A) mode and cryptographic key sizes **128 bit, 256 bit.**

6.1.2.2 SSH Protocol (FCS_SSH_EXT.1)

FCS_SSH_EXT.1.1 The SSH software shall implement the SSH protocol that complies with RFCs 4251, 4252, 4253, 4254 and **RFC 5647, RFC 5656, RFC 6668** as a **client, server**

6.1.2.3 SSH Protocol - Client (FCS_SSHC_EXT.1)

FCS_SSHC_EXT.1.1 The SSH client shall ensure that the SSH protocol implementation supports the following authentication methods as described in RFC 4252: public key-based, and **password-based.**

FCS_SSHC_EXT.1.2 The SSH client shall ensure that, as described in RFC 4253, packets greater than **262144** bytes in an SSH transport connection are dropped.

FCS_SSHC_EXT.1.3 The SSH software shall ensure that the SSH transport implementation uses the following encryption algorithms and rejects all other encryption algorithms: aes128-ctr, aes256-ctr, **aes128-cbc, aes256-cbc, aes128-gcm@openssh.com, aes256-gcm@openssh.com.**

FCS_SSHC_EXT.1.4 The SSH client shall ensure that the SSH transport implementation uses **ssh-rsa, rsa-sha2-256, rsa-sha2-512, ecdsa-sha2-nistp256** and **ecdsa-sha2-nistp384** as its public key algorithm(s) and rejects all other public key algorithms.

Signatures with SHA-1:

The technical decision 578 explicitly discourages signatures with SHA-1 as provided with ssh-rsa. This ST includes this signature type for legacy support only and strongly recommends using one of the other mentioned signature types.

FCS_SSHC_EXT.1.5 The SSH client shall ensure that the SSH transport implementation uses **hmac-sha1, hmac-sha2-256, hmac-sha2-512** and **implicit** as MAC algorithm(s) and rejects all other MAC algorithm(s).

FCS_SSHC_EXT.1.6 The SSH client shall ensure that **diffie-hellman-group14-sha1, ecdh-sha2-nistp256** and **ecdh-sha2-nistp384, ecdh-sha2-nistp521** are the only allowed key exchange methods used for the SSH protocol.

FCS_SSHC_EXT.1.7 The SSH server shall ensure that the SSH connection be rekeyed after **no more than 2^28 packets have been transmitted** using that key.

FCS_SSHC_EXT.1.8 The SSH client shall ensure that the SSH client authenticates the identity of the SSH server using a local database associating each host name with its corresponding public key or **no other methods** as described in RFC 4251 section 4.1.

6.1.2.4 SSH Protocol - Server (FCS_SSHS_EXT.1)

FCS_SSHS_EXT.1.1 The SSH server shall ensure that the SSH protocol implementation supports the following authentication methods as described in RFC 4252: public key-based, and **password-based.**

FCS_SSHS_EXT.1.2 The SSH server shall ensure that, as described in RFC 4253, packets greater than **262144** bytes in an SSH transport connection are dropped.

FCS_SSHS_EXT.1.3 The SSH server shall ensure that the SSH transport implementation uses the following encryption algorithms and rejects all other encryption algorithms: aes128-ctr, aes256-ctr, **aes128-cbc, aes256-cbc, aes128-gcm@openssh.com, aes256-gcm@openssh.com.**

FCS_SSHS_EXT.1.4 The SSH server shall ensure that the SSH transport implementation uses **ssh-rsa, rsa-sha2-256, rsa-sha2-512, ecdsa-sha2-nistp256** and **ecdsa-sha2-nistp384** as its public key algorithm(s) and rejects all other public key algorithms.

Signatures with SHA-1:

The technical decision 578 explicitly discourages signatures with SHA-1 as provided with ssh-rsa. This ST includes this signature type for legacy support only and strongly recommends using one of the other mentioned signature types.

FCS_SSHS_EXT.1.5 The SSH server shall ensure that the SSH transport implementation uses **hmac-sha1, hmac-sha2-256, hmac-sha2-512** and **implicit** as its MAC algorithm(s) and rejects all other MAC algorithm(s).

FCS_SSHS_EXT.1.6 The SSH server shall ensure that **diffie-hellman-group14-sha1, ecdh-sha2-nistp256** and **ecdh-sha2-nistp384, ecdh-sha2-nistp521** are the only allowed key exchange methods used for the SSH protocol.

FCS_SSHS_EXT.1.7 The SSH server shall ensure that the SSH connection be rekeyed after **no more than 2^28 packets have been transmitted** using that key.

6.2 Security Functional Requirements Rationale

6.2.1 Coverage

The following table provides a mapping of SFR to the security objectives, showing that each security functional requirement addresses at least one security objective.

Security functional requirements	Objectives
FCS_CKM.1	O.PROTECTED_COMMS
FCS_CKM.2	O.PROTECTED_COMMS
FCS_CKM_EXT.4	O.PROTECTED_COMMS
FCS_COP.1(1)	O.PROTECTED_COMMS, O.PROTECTED_STORAGE
FCS_COP.1(2)	O.INTEGRITY, O.PROTECTED_COMMS
FCS_COP.1(3)	O.INTEGRITY, O.PROTECTED_COMMS
FCS_COP.1(4)	O.INTEGRITY, O.PROTECTED_COMMS
FCS_RBG_EXT.1	O.PROTECTED_COMMS, O.PROTECTED_STORAGE
FCS_STO_EXT.1	O.PROTECTED_STORAGE
FCS_TLSC_EXT.1	O.PROTECTED_COMMS
FCS_TLSC_EXT.2	O.PROTECTED_COMMS
FDP_ACF_EXT.1	O.PROTECTED_STORAGE
FMT_MOF_EXT.1	O.MANAGEMENT
FMT_SMF_EXT.1	O.MANAGEMENT
FPT_ACF_EXT.1	O.INTEGRITY
FPT_ASLR_EXT.1	O.INTEGRITY
FPT_SBOP_EXT.1	O.INTEGRITY
FPT_TST_EXT.1	O.INTEGRITY
FPT_TUD_EXT.1	O.INTEGRITY
FPT_TUD_EXT.2	O.INTEGRITY
FAU_GEN.1	O.ACCOUNTABILITY

Security functional requirements	Objectives
FIA_AFL.1	O.INTEGRITY
FIA_UAU.5	O.INTEGRITY
FIA_X509_EXT.1	O.INTEGRITY, O.PROTECTED_COMMS
FIA_X509_EXT.2	O.PROTECTED_COMMS
FTP_ITC_EXT.1	O.ACCOUNTABILITY, O.INTEGRITY, O.PROTECTED_COMMS
FTP_TRP.1	O.MANAGEMENT
FCS_COP.1(5)	O.INTEGRITY
FCS_SSH_EXT.1	O.PROTECTED_COMMS
FCS_SSHC_EXT.1	O.PROTECTED_COMMS
FCS_SSHS_EXT.1	O.PROTECTED_COMMS

Table 8: Mapping of security functional requirements to security objectives

6.2.2 Sufficiency

The following rationale provides justification for each security objective for the TOE, showing that the security functional requirements are suitable to meet and achieve the security objectives.

Security objectives	Rationale
O.ACCOUNTABILITY	FAU_GEN.1 defines the auditable events that must be generated to diagnose the cause of unexpected system behavior. FTP_ITC_EXT.1 provides a mechanism for the TSF to transmit the audit data to a remote system.
O.INTEGRITY	FPT_SBOP_EXT.1 enforces stack buffer overflow protection that makes it more difficult to exploit running code. FPT_ASLR_EXT.1 prevents attackers from exploiting code that executes in static known memory locations. FPT_TUD_EXT.1 and FPT_TUD_EXT.2 enforce integrity of software updates. FCS_COP.1(2), FCS_COP.1(3), and FCS_COP.1(4) provide the cryptographic mechanisms that are used to verify integrity values. FPT_ACF_EXT.1 guarantees the integrity of critical components by preventing unauthorized modifications of them. FPT_X509_EXT.1 provides X.509 certificates as a way of validating software integrity. FPT_TST_EXT.1 verifies the integrity of stored code. FIA_UAU.5 provides mechanisms that prevent untrusted users from accessing the TSF and FIA_AFL.1 prevents brute-force authentication attempts. FTP_ITC_EXT.1 provides trusted remote communications which makes a remote authenticated session less susceptible to compromise. FCS_COP.1(5) supports the SSH protocol implementation.
O.MANAGEMENT	FMT_SMF_EXT.1 defines the TOE's management functions and FMT_MOF_EXT.1 defines the privileges required to invoke them. FTP_TRP.1 provides one or more secure remote interfaces for management of the TSF.

Security objectives	Rationale
O.PROTECTED_STORAGE	Rationale: FCS_STO_EXT.1 provides a mechanism by which the TOE can designate data as 'sensitive' and subsequently require it to be encrypted. FCS_COP.1(1) defines the symmetric algorithm used to encrypt and decrypt sensitive data. FCS_RBG_EXT.1 defines the random bit generator used to create the symmetric keys used to perform this encryption and decryption. FDP_ACF_EXT.1 enforces logical access control on stored data.
O.PROTECTED_COMMS	FCS_TLSC_EXT.1, FCS_TLSC_EXT.2 define the ability of the TOE to act as a TLS client as a method of enforcing protected communications. FCS_CKM.1, FCS_CKM.2, FCS_COP.1(1), FCS_COP.1(2), FCS_COP.1(3), FCS_COP.1(4), and FCS_RBG_EXT.1 define the cryptographic operations and key lifecycle activity used to support the establishment of protected communications. FIA_X509_EXT.1 defines how the TSF validates x.509 certificates as part of establishing protected communications. FIA_X509_EXT.2 defines the trusted communication protocols for which the TOE must perform certificate validation operations. FTP_ITC_EXT.1 defines the trusted communications channels supported by the TOE. The SSH protocol is defined by FCS_SSH_EXT.1, FCS_SSHC_EXT.1, and FCS_SSHS_EXT.1.

Table 9: Security objectives for the TOE rationale

6.2.3 Security requirements dependency analysis

[OSPP] and [SSHEP] are evaluated PPs and this document contains all requirements from [OSPP] and [SSHEP]. Therefore, the dependencies are not applicable as the PP and EP are approved.

6.3 Security Assurance Requirements

The security assurance requirements for the TOE are by [OSPP].

The security assurance requirements (SARs) for the TOE are defined in the [OSPP] protection profile.

The following table shows the SARs, and the operations performed on the components according to CC part 3: iteration (Iter.), refinement (Ref.), assignment (Ass.) and selection (Sel.).

Security assurance class	Security assurance requirement	Source	Operations			
			Iter.	Ref.	Ass.	Sel.
ASE Security Target evaluation	ASE_CCL.1 Conformance claims	OSPP	No	No	No	No
	ASE_ECD.1 Extended components definition	OSPP	No	No	No	No
	ASE_INT.1 ST introduction	OSPP	No	No	No	No
	ASE_OBJ.2 Security objectives	OSPP	No	No	No	No
	ASE_REQ.2 Derived security requirements	OSPP	No	No	No	No
	ASE_SPD.1 Security problem definition	OSPP	No	No	No	No
	ASE_TSS.1 TOE summary specification	OSPP	No	No	No	No

Security assurance class	Security assurance requirement	Source	Operations			
			Iter.	Ref.	Ass.	Sel.
ADV Development	ADV_FSP.1 Basic functional specification	OSPP	No	No	No	No
AGD Guidance documents	AGD_OPE.1 Operational user guidance	OSPP	No	No	No	No
	AGD_PRE.1 Preparative procedures	OSPP	No	No	No	No
ALC Life-cycle support	ALC_CMC.1 Labelling of the TOE	OSPP	No	No	No	No
	ALC_CMS.1 TOE CM coverage	OSPP	No	No	No	No
	ALC_TSU_EXT.1	OSPP	No	No	No	No
ATE Tests	ATE_IND.1 Independent testing - conformance	OSPP	No	No	No	No
AVA Vulnerability assessment	AVA_VAN.1 Vulnerability survey	OSPP	No	No	No	No

Table 10: SARs

6.4 Security Assurance Requirements Rationale

The rationale for the chosen SARs is provided in [OSPP].

7 TOE Summary Specification

7.1 TOE Security Functionality

The following section explains how the security functions are implemented. The different TOE security functions cover the various SFR classes.

The primary security features of the TOE are:

- Audit
- Cryptographic services
- Identification and Authentication
- Discretionary Access Control
- Security Management

7.1.1 Audit

The Lightweight Audit Framework (LAF) is designed to be an audit system for Linux compliant with the requirements from Common Criteria. LAF is able to intercept all system calls as well as retrieving audit log entries from privileged user space applications. The subsystem allows configuring the events to be actually audited from the set of all events that are possible to be audited. Those events are configured in a specific configuration file and then the kernel is notified to build its own internal structure for the events to be audited.

7.1.1.1 Audit functionality

The Linux kernel implements the core of the LAF functionality. It gathers all audit events, analyzes these events based on the audit rules and forwards the audit events that are requested to be audited to the audit daemon executing in user space.

Audit events are generated in various places of the kernel. In addition, a user space application can create audit records which needs to be fed to the kernel for further processing.

The audit functionality of the Linux kernel is configured by user space applications which communicate with the kernel using a specific netlink communication channel. This netlink channel is also to be used by applications that want to send an audit event to the kernel.

The kernel netlink interface is usable only by applications possessing the following capabilities:

- CAP_AUDIT_CONTROL: Performing management operations like adding or deleting audit rules, setting or getting auditing parameters;
- CAP_AUDIT_WRITE: Submitting audit records to the kernel which in turn forwards the audit records to the audit daemon.

Based on the audit rules, the kernel decides whether an audit event is discarded or to be sent to the user space audit daemon for storing it in the audit trail. The kernel sends the message to the audit daemon again using the above mentioned netlink communication channel. The audit daemon writes the audit records to the audit trail. An internal queuing mechanism is used for this purpose. When the queue does not have sufficient space to hold an audit record the TOE switches into single user mode, is halted, all processes are stopped that generate audit records, or the audit daemon executes an administrator-specified notification action depending on the configuration of the audit daemon. This ensures that audit records do not get lost due to resource shortage and the administrator can backup and clear the audit trail to free disk space for new audit logs.

Access to audit data by normal users is prohibited by the discretionary access control function of the TOE, which is used to restrict the access to the audit trail and audit configuration files to the system administrator only.

The system administrator can define the events to be audited from the overall events that the Lightweight Audit Framework using simple filter expressions. This allows for a flexible definition of the events to be audited and the conditions under which events are audited. The system administrator is also able to define a set of user IDs for which auditing is active or alternatively a set of user IDs that are not audited.

The system administrator can select files to be audited by adding them to a watch list that is loaded into the kernel.

The audit trail is stored in files that are readable by the root user only.

7.1.1.2 Audit trail

An audit record consists of one or more lines of text containing fields in a “keyword=value” tagged format. The following information is contained in all audit record lines:

- Type: indicates the source of the event, such as SYSCALL, PATH, USER_LOGIN, or LOGIN
- Timestamp: Date and time the audit record was generated
- Audit ID: unique numerical event identifier
- Login ID (“auid”), the user ID of the user authenticated by the system (regardless if the user has changed his real and / or effective user ID afterwards)
- Effective user ID: the effective user ID of the process at the time the audit event was generated
- Success or failure (where appropriate)
- Process ID of the subject that caused the event (PID)

This information is followed by event specific data. In some cases, such as SYSCALL event records involving file system objects, multiple text lines will be generated for a single event, these all have the same time stamp and audit ID to permit easy correlation.

The audit trail is stored in ASCII text. The TOE provides tools for managing ASCII files that can be used for post-processing of audit data. These tools include:

- less - reads the ASCII audit data
- ausearch - allows selective extraction of records from the audit trail using defined selection criteria
- sort - The audit records are listed in chronological order by default. The sort utility can be used together with ausearch to use a different sorting order.

The audit trail is stored in files which are accessible by root only.

7.1.1.3 Centralized audit collection and management

The audit daemon of the TOE that stores the local audit trail is capable of remote auditing. The concept of remote auditing implies that one machine - the client - does not maintain the audit trail locally, but sends it to an audit server that centrally collects the audit data. The functionality of remote auditing contains the following two aspects implemented by the audit daemon:

- Audit server: The audit daemon can be configured to listen on a network port for other audit daemons to submit audit data. The communication is protected against eavesdropping using GSSAPI. Different protection mechanisms can be employed using GSSAPI, like Kerberos or SPNEGO (note: none of the security mechanisms is part of the evaluation). The audit daemon receives any audit records and stores them as part of the local audit trail together with the information about the source of the audit trail.
- Audit client: The audit daemon can be configured to send the audit trail off to an audit server via the network instead of storing the data locally.

The TOE is capable of providing the server side as well as the client side for remote auditing. For the client side, the TOE can always be configured to store the audit trail local to the client or to send it off to the server. Therefore, the TOE is always able to provide a self-sufficient audit trail if configured by the authorized administrator.

In conjunction with the centralized storage of the audit trail, the audit system of the TOE can also be administered centrally. The central management depends on using SSH to access the auditctl command on all administered servers:

- the administrator can either configure the audit functionality locally and use the scp tool to copy it to the remote target systems, or
- the administrator can configure the audit functionality directly on the remote systems using the ssh tool.

The remote management allows the configuration of all audit-related aspects on the remote system in the same way as the local audit system can be configured.

7.1.2 Cryptographic services

The TOE provides cryptographically secured network communication channels to allow remote users to interact with the TOE. Using one of the following cryptographically secured network channels, a user can request the following services:

- OpenSSH: The OpenSSH application provides access to the command line interface of the TOE. Users may employ OpenSSH for interactive sessions as well as for non-interactive sessions. The console provided via OpenSSH provides the same environment as a local console. OpenSSH implements the SSHv2 protocol.
- TLS: The OpenSSL library suite implements the TLS protocol family.

In addition to the cryptographically secured communication channels, the TOE also provides cryptographic algorithms for general use.

The cryptographic primitives for implementing the above mentioned cryptographic communication protocols are provided by OpenSSL.

7.1.2.1 SSHv2 Protocol

The TOE provides the Secure Shell Protocol Version 2 (SSH v2.0) to allow users from a remote host to establish a secure connection and perform a logon to the TOE.

The following table documents implementation details concerning the OpenSSH implementation’s compliance to the relevant standards. It addresses areas where the standards permit different implementation choices such as optional features.

Reference	**Description**	**Implementation Details**
RFC 4253 chapter 5	Compatibility with old SSH versions	The OpenSSH implementation is capable of interoperating with clients and servers using the old 1.x protocol. That functionality is explicitly disabled in the evaluated configuration, it permits protocol version 2.0 exclusively.
RFC 4253 section 6.2	Compression	OpenSSH supports the OPTIONAL "zlib" compression method.
RFC 4253 section 6.3	Encryption	The ciphers supported in the evaluated configuration are listed in FCS_SSHC_EXT.1 and FCS_SSHS_EXT.1 for the SSH protocol.

Reference	Description	Implementation Details
RFC 4252 chapter 7	Public Key Authentication Method: "publickey"	The ciphers supported in the evaluated configuration are listed in FCS_SSHC_EXT.1 and FCS_SSHS_EXT.1 for the SSH protocol. This REQUIRED authentication method is supported by OpenSSH but can be disabled by the administrator of the OpenSSH daemon.
RFC 4252 chapter 8	Password Authentication Method: "password"	This SHOULD authentication method is supported by OpenSSH but can be disabled by the administrator of the OpenSSH daemon.
RFC 4252 chapter 8	Password change request and setting new password	The OpenSSH implementation supports the optional password change mechanism in the evaluated configuration.
RFC 4252 chapter 9	Host-Based Authentication: "hostbased"	This OPTIONAL authentication method is disabled in the evaluated configuration.

Table 11: SSH implementation notes

The TOE supports the generation of RSA, as well as ECDSA key pairs. These key pairs are used by OpenSSH for the host keys as well as for the per-user keys. When a user registers his public key with the user he wants to access on the server side, a key-based authentication can be performed instead of a password-based authentication. The key generation mechanism uses the random number generator of the underlying cryptographic library. The evaluated configuration permits the import of externally-generated key pairs.

The TOE supports the following security functions of the SSH v2.0 protocol:

- Establishing a secure communication channel using the following cryptographic functions provided by the SSH v2.0 protocol:
  - Encryption as defined in section 4.3 of [RFC4253] - the keys are generated using the random number generator of the underlying cryptographic library;
  - Diffie-Hellman key agreement as defined in section 6.1 of [RFC4253];
  - The keyed hash function for integrity protection as defined in section 4.4 of [RFC4253].

  Note: The protocol supports more cryptographic algorithms than the ones listed above. Those other algorithms are not covered by this evaluation and should be disabled or not used when running the evaluated configuration.
- Performing user authentication using the standard password-based authentication method the TOE provides for users (password authentication method as defined in chapter 5 of [RFC4252]).
- Performing user authentication using a RSA, or ECDSA key-based authentication method (public key authentication method as defined in chapter 5 of [RFC4252]).
- Checking the integrity of the messages exchanged and close down the connection in case an integrity error is detected.

The OpenSSH applications of sshd, ssh and ssh-keygen use the OpenSSL random number generator seeded by the getrandom system call to generate cryptographic keys. OpenSSL provides different DRNGs depending whether the FIPS 140-2 mode is enabled in the system. Yet, the evaluated configuration only allows the FIPS 140-2 mode to be set.

The cryptographic implementations ensure that sensitive data is appropriately zeroized before releasing the associated memory.

7.1.2.2 Confidentiality protected data storage

File system objects are stored on block devices, such as partitions on hard disk. The Linux operating systems offers the use of an additional layer between the file systems and the physical block device to encrypt and decrypt any data transmitted between the file system and the block device. The dm_crypt functionality uses the Linux device mapper to provide such encryption and decryption operation that is transparent to the file system and therefore to the user.

Before mounting the block device that is protected by the dm_crypt encryption scheme, the owner of the encrypted block device has to provide a passphrase. This passphrase is used to decrypt the symmetric volume key which is injected into the kernel. Using that volume key, the kernel is now able to decrypt (to unlock) the data on the device and provides access to data stored on that device. At this point, the file system can be mounted as the file system can now be read.

Once the dm_crypt protected device is unlocked and mounted, it is accessible as any other file system. When it is unmounted and locked (i.e. the kernel is informed to discard the volume key), all data on the device is inaccessible. Even administrative users like the root user are not able to access any data any more. When an administrator would access the raw hardware hosting the block device, only encrypted data can be read.

For the cryptographic operation, the creator of the dm_crypt block device can select the cipher. When creating the dm_crypt block device, the volume key is obtained from the Linux random number generator and stored on the block device encrypted with the user's passphrase. The key derivation mechanism from the user's password is based on the LUKS mechanism.

The encryption and decryption operation of the device is implemented by the kernel. To unlock the encrypted volume key stored on the protected block device, the cryptsetup application performs the following steps:

1. obtain the user's passphrase
2. apply the LUKS key derivation mechanism on the passphrase
3. read the encrypted volume key from the device
4. decrypt the volume key with the key derived from the user's passphrase
5. inject the decrypted volume key into the kernel and set up the mapping between the block device and the volume key

Using the cryptsetup tool, the volume key can also be transferred by encrypting it with another passphrase which can be given to another user. The transfer follows the same steps outlined for the unlocking operation, but instead of injecting the decrypted volume key into the kernel, cryptsetup fetches the new passphrase from the user, applies the LUKS mechanism on that passphrase, encrypts the volume key with the derived key and stores the encrypted volume key in a separate area on the device. At this point, the volume key is now stored encrypted in two separate places.

Similarly, the cryptsetup tool can be used to erase the storage location of one encrypted volume key which implies that the user owning the passphrase of the affected encrypted volume key is not able to unlock the block device any more.

In FIPS 140-2 mode, the libgcrypt DRBG is used for generating keys seeded by the getrandom system call.

The installer with the exception of IBM Z System allows the configuration of the full disk encryption schema where the entire disk is protected, except the /boot partition.

7.1.2.3 TLSv1.2 Protocol

The TOE provides TLSv1.2 to allow users from a remote host to establish a secure channel to the TOE. In contrast to the SSH protocol described above, the TLS protocol performs the support authentication as part by verifying the RSA certificates. The TOE can be configured using a bi-directional certificate verification where the server side (implemented by the libvirt daemon) validates the client certificate.

The following table documents implementation details concerning the OpenSSL implementation's compliance to the relevant standards. It addresses areas where the standards permit different implementation choices such as optional features.

Reference	**Description**	**Implementation Details**
RFC 5246 section 7.3	Handshake protocol overview: certificates	The evaluated configuration always uses server certificates. Client certificates are used to allow the server to authenticate the client.
RFC 5246 appendix F.1.1.2	Temporary RSA keys	Not applicable, the evaluated configuration does not limit the size of encryption keys to 512 bits.
RFC 5246 appendix D.1	Random Number Generation and Seeding	OpenSSL uses data from the Linux kernel random number generator, a persistent entropy pool file, and volatile system statistics to seed the PRNG.
RFC 5246 appendix D.2	Certificates and authentication	The evaluated configuration supports verification of certificate chains.
RFC 5246 appendix D.3	Cipher suites	The ciphers supported in the evaluated configuration are listed in FCS_TLSC_EXT.1 for the TLS protocol.
RFC 5246 appendix E	SSLv2, SSLv3, TLSv1.0, TLSv1.1 Backward Combatibility	The OpenSSL implementation supports the backwards compatible protocol, but this is disabled in the evaluated configuration. It permits use of TLSv1.2 exclusively.

Table 12: TLS implementation notes

The TOE supports the generation of the RSA key pair used by the server. The key generation mechanism uses the Linux kernel random number generator. The evaluated configuration also allows the use of an externally-generated certificate. A widely accepted Certification Authority might be used to generate and/or sign such a certificate (allowing a wide community trusting this CA to validate the certificate). In a closed community it might also be sufficient to have one server within the community to act as a CA. The OpenSSL library provides the functions to set up such a CA, but those functions are not subject of this Security Target.

The TLS protocol implementation of the TOE complies with RFC 5246 page 60 to counter the Bleichenbacher Attack by sending the TLS finished message instead of informing about the failed pre-master decryption operation.

The key material used for by TLS is obtained from the getrandom system call and is fed into the state random number generator.

7.1.3 Identification and Authentication

User identification and authentication in the TOE includes all forms of interactive login (e.g. using the SSH protocol or log in at the local console) as well as identity changes through the su and sudo commands. These all rely on explicit authentication information provided interactively by a user. In addition, the key-based authentication mechanism of the OpenSSH server is another form of of authentication.

7.1.3.1 PAM-based identification and authentication mechanisms

Linux uses a suite of libraries called the "Pluggable Authentication Modules" (PAM) that allow an administrative user to choose how PAM-aware applications authenticate users. The TOE provides PAM modules that implement all the security functionality to:

- Provides login control and establishing all UIDs, GIDs and login ID for a subject
- Ensure the quality of passwords
- Enforce limits for accounts (such as the number of maximum concurrent sessions allowed for a user)
- Enforce the change of passwords after a configured time including the password quality enforcement
- Enforcement of locking of accounts after failed login attempts.
- Restriction of the use of the root account to certain terminals
- Restriction of the use of the su and sudo commands

The login processing sets the real, file system effective and login UID as well as the real, effective, file system GID and the set of supplemental GIDs of the subject that is created. It is of course up to the client application usually provided by a remote system to protect the user’s entry of a password correctly (e. g. provide only obscured feedback).

During login processing, the user is shown a banner. After successful authentication, the login time is recorded.

After a successful identification and authentication, the TOE initiates a session for the user and spawns the initial login shell as the first process the user can interact with. The TOE provides a mechanism to lock a session either automatically after a configurable period of inactivity for that session or upon the user's request.

User accounts are stored in configuration files (/etc/passwd and /etc/shadow). Both are writable to the root user only. In addition, /etc/shadow is readable to the root user only. Modification of both files is performed using a set of administrative applications. When a user ID is removed, its entry in the configuration files is removed by the administrative interfaces. Therefore, a log-in using such a removed user ID is unsuccessful.

7.1.3.2 User Identity Changing

Users can change their identity (i.e., switch to another identity) using one of the following commands provided with the TOE:

su command

The su command is intended for a switch to a another identity that establishes a new login session and spawns a new shell with the new identity. When invoking su, the user must provide the credentials associated with the target identity - i.e. when the user wants to switch to another user ID, it has to provide the password protecting the account of the target user.

The primary use of the su command within the TOE is to allow appropriately authorized individuals the ability to assume the root identity to perform administrative actions. In this system the capability to login as the root identity has been restricted to defined terminals only. In addition the use of the su command to switch to root has been restricted to users belonging to a special group. Users that don’t have access to a terminal where root login is allowed and are not member of that special group will not be able to switch their real, file system and effective user ID to root even if they would know the authentication information for root. Note that when a user executes a program that has the setuid bit set, only the effective user ID and file system ID are changed to that of the owner of the file containing the program while the real user ID remains that of the caller. The login ID is neither changed by the su command nor by executing a program that has the setuid or setgid bit set as it is used for auditing purposes.

sudo command

The sudo command is intended for giving users permissions to execute commands with another user identity. When invoking sudo, the user has to authenticate with this credentials.

Sudo is associated with sophisticated ruleset that can be engaged to specify which:

- source user ID
- originating from which host
- can access a command, a command with specific configuration flags, or all commands within a directory
- with which new user identity.

When switching identities, the real, file system, and effective user ID, and real, file system, and effective group ID are changed to the one of the user specified in the command (after successful authentication as this user).

Note: The login ID is not retained for the following special case:

1. User A logs into the system.
2. User A uses su to change to user B.
3. User B now edits the cron or at job queue to add new jobs. This operation is appropriately audited with the proper login ID.
4. Now when the new jobs are executed as user B, the system does not provide the audit information that the jobs are created by user A.

The su command invokes the common authentication mechanism to validate the supplied authentication.

7.1.3.3 Authentication Data Management

Each TOE instance maintains its own set of users with their passwords and attributes. Although the same human user may have accounts on different servers interconnected by a network and running an instantiation of the TOE, those accounts and their parameter are not synchronized on different TOE instances. As a result the same user may have different user names, different user Ids, different passwords and different attributes on different machines within the networked environment. Existing mechanism for synchronizing this within the whole networked system are not subject to this evaluation.

Each TOE instance within the network maintains its own administrative database by making all administrative changes on the local TOE instance. System administration has to ensure that all machines within the network are configured in accordance with the requirements defined in this Security Target.

The file /etc/passwd contains for each user the user’s name, the id of the user, an indicator whether the password of the user is valid, the principal group id of the user and other (not security relevant) information. The file /etc/shadow contains for each user a hash of the user's password, the userid, the time the password was last changed, the expiration time as well as the validity period of the password and some other information that are not subject to the security functions as defined in this Security Target. Users are allowed to change their passwords by using the passwd command. This application is able to read and modify the contents of /etc/shadow for the user’s password entry, which would ordinarily be inaccessible to a non-privileged user process. Users are also warned to change their passwords at login time if the password will expire soon, and are prevented from logging in if the password has expired.

The time of the last successful logins is recorded in the directory /var/log/tallylog where one file per user is kept.

The TOE displays informative banners before or during the login to users. The banners can be specified with the files /etc/issue for log ins via the physical console or /etc/issue.net for remote log ins, such as via SSH. When performing a log in on the physical console, the banner is displayed above the username and password prompt. For log ins via SSH, the banner is displayed to the remote peer during the SSH-session handshake takes place. The remote SSH client will display the banner to the user. When using the provided OpenSSH client, the banner is displayed when the user instructs the OpenSSH client to log into the remote system.

7.1.3.4 SSH key-based authentication

In addition to the PAM-based authentication outlined above, the OpenSSH server is able to perform a key-based authentication. When a user wants to log on, instead of providing a password, the user applies his SSH key. After a successful verification, the OpenSSH server considers the user as authenticated and performs the PAM-based operations as outlined above.

To establish a key-based authentication, a user first has to generate an RSA, or ECDSA key pair. The private part of the key pair remains on the client side. The public part is copied to the server into the file .ssh/authorized_keys which resides in the home directory of the user he wants to log on as. When the login operation is performed the SSHv2 protocol tries to perform the "publickey" authentication using the private key on the client side and the public key found on the server side. The operations performed during the publickey authentication is defined in [RFC4252]☝ chapter 7.

Users have to protect their private key part the same way as protecting a password. Appropriate permission settings on the file holding the private key is necessary. To strengthen the protection of the private key, the user can encrypt the key where a password serves as key for the encryption operation. See ssh-keygen(1) for more information.

7.1.3.5 Session locking

The TOE uses the screen(1) application which locks the current session of the user either after an administrator-specified time of inactivity or upon the user's request.

To unlock the session, the user must supply his password. Screen uses PAM to validate the password and allows the user to access his session after a successful validation.

7.1.4 Discretionary Access Control

The general policy enforced is that subjects (i.e., processes) are allowed only the accesses specified by the policies applicable to the object the subject requests access to. Further, the ability to propagate access permissions is limited to those subjects who have that permission, as determined by the policies applicable to the object the subject requests access to.

A subject may possess one or more of the following capabilities which provide the following exemptions from the DAC mechanism:

- CAP_DAC_OVERRIDE: A process with this capability is exempt from all restrictions of the discretionary access control and can perform any action desired. For the execution of a file, the permission bit vector of that file must contain at least one execute bit.
- CAP_DAC_READ_SEARCH: A process with this capability overrides all DAC restrictions regarding read and search on files and directories.
- CAP_CHOWN: A process with this capability is allowed to make arbitrary changes to a file's UID or GID.
- CAP_FOWNER: Setting permissions and ownership on objects even if the process' UID does not match the UID of the object.

- CAP_FSETID: Don't clear SUID and SGID permission bits when a file is modified.

DAC provides the mechanism that allows users to specify and control access to objects that they own. DAC attributes are assigned to objects at creation time and remain in effect until the object is destroyed or the object attributes are changed. DAC attributes exist for, and are particular to, each type of named object known to the TOE. DAC is implemented with permission bits and, when specified, ACLs.

The outlined DAC mechanism applies only to named objects which can be used to store or transmit user data. Other named objects are also covered by the DAC mechanism but may be supplemented by further restrictions. These additional restrictions are out of scope for this evaluation. Examples of objects which are accessible to users that cannot be used to store or transmit user data are: virtual file systems externalizing kernel data structures (such as most of procfs, sysfs, binfmt_misc) and process signals.

During creation of objects, the TSF ensures that all residual contents is removed from that object before making it accessible to the subject requesting the creation.

When data is imported into the TOE (such as when mounting disks created by other trusted systems), the TOE enforces the permission bits and ACLs applied to the file system objects.

7.1.4.1 Permission bits

The TOE supports standard UNIX permission bits to provide one form of DAC for file system objects in all supported file systems. There are three sets of three bits that define access for three categories of users: the owning user, users in the owning group, and other users. The three bits in each set indicate the access permissions granted to each user category: one bit for read (r), one for write (w) and one for execute (x). Note that write access to file systems mounted as read only (e. g. CD-ROM) is always rejected (the exceptions are character and block device files which can still be written to as write operations do not modify the information on the storage media). The SVTX (sticky) attribute is used for world-writeable temp directories preventing the removal of files by users other than the owner.

Each process has an inheritable “umask” attribute which is used to determine the default access permissions for new objects. It is a bit mask of the user/group/other read/write/execute bits, and specifies the access bits to be removed from new objects. For example, setting the umask to “002” ensures that new objects will be writable by the owner and group, but not by others. The umask is defined by the administrator in the /etc/login.defs file or 022 by default if not specified.

When using the VFAT file system at the /boot/efi mount point, the permissions applicable for every file system object beneath that mount point is specified with mount options. The mount system call allows the specification of the owning ID, group ID and the permissions applicable to the file system objects. The permissions specified at the mount point have the same semantics as the Unix permission bits.

Modification of DAC attributes is restricted to the owner of the object or users with the aforementioned capabilities.

7.1.4.2 Access Control Lists (ACLs)

The TOE provides support for POSIX type ACLs to define a fine grained access control on a user basis. ACLs are supported for all file system objects stored with the following file systems:

- btrfs
- ext3
- ext4
- xfs
- tmpfs

An ACL entry contains the following information:

- A tag type that specifies the type of the ACL entry
- A qualifier that specifies an instance of an ACL entry type
- A permission set that specifies the discretionary access rights for processes identified by the tag type and qualifier

An ACL contains exactly one entry of three different tag types (called the "required ACL entries” forming the "minimum ACL"). The standard UNIX file permission bits as described in the previous section are represented by the entries in the minimum ACL.

A default ACL is an additional ACL which may be associated with a directory. This default ACL has no effect on the access to this directory. Instead the default ACL is used to initialize the ACL for any file that is created in this directory. If the new file created is a directory it inherits the default ACL from its parent directory. When an object is created within a directory and the ACL is not defined with the function creating the object, the new object inherits the default ACL of its parent directory as its initial ACL.

Modification of DAC attributes is restricted to the owner of the object or users with the aforementioned capabilities.

7.1.4.3 File system objects

File system objects access checks are performed when the object is initially opened, and are not checked on each subsequent access. Changes to access controls (i.e., revocation) are effective with the next attempt to open the object.

7.1.4.4 IPC objects

The TOE implements the following standard types of IPC mechanisms:

- SYSV Shared Memory
- SYSV and POSIX Message Queues
- SYSV Semaphores

Access to the above mentioned IPC mechanisms are governed by UNIX permission bits.

As the IPC objects of UNIX domain socket special files and Named Pipes are represented as file system objects, the access control mechanism covering file system objects are applicable to these IPC mechanisms too.

The TOE maintains IPC object types where each process has its own namespace for that object type: sockets - including network sockets. Access to the socket is only possible by the process whose socket namespace contains the socket reference. Setting of permissions for such objects can be handled using file descriptor passing.

Modification of DAC attributes is restricted to the owner of the object or users with the aforementioned capabilities.

7.1.4.5 at and cron jobs queues

at and cron jobs can only be accessed (read/added/modified/deleted) by the owning user. The TOE maintains at and cron job queues for each user.

The root user can always access every at or cron job queue.

The at or cron jobs are started with the UIDs/GIDs of the creator of the job.

7.1.5 Security Management

The security management facilities provided by the TOE are usable by authorized users and/or authorized administrators to modify the configuration of TSF. The configuration of TSF are hosted in the following locations:

- Configuration files (or TSF databases)
- Data structures maintained by the kernel and within the kernel memory

The TOE provides applications to authorized users as well as authorized administrators to perform various administrative tasks. These applications are documented as part of the administrator and user guidance. These applications are either used to modify configuration files or to access parameters controlled and enforced by the kernel via kernel-provided interfaces to user space.

Configuration options are stored in different configuration files. These files are protected using the DAC mechanisms against unauthorized access where usually the root user only is allowed to write to the files. In some special cases (like for /etc/shadow), the file is even readable to the root user only. It is the task of the persons responsible for setting up and administrating the system to ensure that the access control features of the TOE are used throughout the lifetime of the system to protect those databases. These configuration files are accessed using applications which are able to interpret the contents of these configuration files. Each TOE instance maintains its own TSF database. Synchronizing those databases is not performed in the evaluated configuration. If such synchronization is required by an organization it is the responsibility of an administrative user of the TOE to achieve this either manually or with some automated assistance.

To access data structures maintained by the kernel, applications use the kernel-provided interfaces, such as system calls, virtual file systems, netlink sockets, and device files. These kernel interfaces are restricted to authorized administrators or authorized users, if applicable, by either using DAC (for virtual file system objects) or special kernel-internal verification checks for each interface.

7.1.5.1 Privileges

Privileges to perform administrative actions are maintained by the TOE. These privileges are separated into privileges to act on data or access functionality in user space and in kernel space.

Functionality accessible in user space are applications that can be invoked by users. Also, data accessible in user space is either data maintained with an application or data stored in persistent or transient storage objects. Privileges are controlled by permissions to invoke applications and to access data. For example, the configuration files including the user databases of /etc/passwd and /etc/shadow are accessible to the root user only. Therefore, the root user is given the privilege to perform modifications on this configuration data which constitutes administrative actions.

Functionality and data maintained by the kernel must be accessed using system calls. The kernel implements a privilege check for functions and data that shall not be accessible by normal users. These privileges are controlled with capabilities that can be assigned to processes. If a process is assigned with a capability, it is allowed to request special operations that other processes cannot. To implement consistency with the Unix legacy, processes with the effective UID of zero are implicitly given all capabilities. However, these processes may decide to drop capabilities. Such capabilities are marked by names with the prefix of "CAP_" throughout this document. The Linux kernel implements many more capabilities than mentioned in this document. These unmentioned capabilities protect functions that do not directly cover SFR functionality but need to be protected to ensure the integrity of the system and its resources.

7.1.5.2 Approval and delegation of management functions

Using the sudo command, authorized administrators can approve that other users can perform management tasks. Once the administrator approves the operation, the /etc/sudoers file is modified to grant the user the right to perform the administrative operation.

Using the /etc/sudoers file, the administrator can specify the approval rules based on the following fine-grained properties:

- Specification of the command that can be executed. The command may contain wild cards.
- Specification of the target user ID or group ID the command shall be executed with.
- Specification of the user ID or group ID (where all members of the group are covered) which are allowed by this rule.

Using the sudo command and the associated /etc/sudoers configuration file, the administrative users, i.e. the users allowed to use the root UID are allowed to delegate parts or all of their authority to other users.

7.1.6 Cryptographic key management

The TOE supports the generation of RSA, and ECDSA keys for the OpenSSH host key as well as the OpenSSH user keys using the ssh-keygen(1) application. The following key sizes are supported:

- RSA: 2048 bits, 3072 bits, 4096 bits, 6144 bits, 8192 bits
- ECDSA: key sizes as defined by the selected curve.

The TOE supports the generation of RSA, and ECDSA keys for the TLS protocol using the openssl(1) application. The following key sizes are supported:

- RSA: any key size that is a multiple of a byte between 2048 and 32768 bits
- ECDSA: key sizes as defined by the selected curve.

The support for FFC DSS key generation is used to support the FFC-DH key agreement for both, SSH and TLS. Key sizes of 2048 bits, 3072 bits, 4096 bits, 6144 bits and 8192 bits are supported compliant to the selected domain parameter group.

For details, see Cryptographic services.

This supports FCS_CKM.1.

7.1.7 Cryptographic key establishment

The key establishment using FFC-Diffie Hellman as well as ECC-Diffie Hellman is supported for both, TLS and SSH.

For details, see Cryptographic services.

This supports FCS_CKM.2.

7.1.8 Cryptographic key destruction

Ephemeral cryptographic key material is held in RAM and is zeroized as soon as it is not required any more by overwriting the memory location with zeros.

The following keys are managed by OpenSSL on behalf of TLS and SSH:

- Ephemeral symmetric keys: The keys are derived during the TLS/SSH handshake from the Diffie-Hellman operation and stay in memory until rekey or destruction of the connection.
- Ephemeral MAC keys: The keys are derived during the TLS/SSH handshake from the Diffie-Hellman operation and stay in memory until rekey or destruction of the connection.
- Ephemeral DH/ECDH keys: The keys are generated by OpenSSL using its DRBG during the TLS/SSH handshake from the Diffie-Hellman operation and stay in memory until the handshake is completed.

- Ephemeral representation of asymmetric keys: The keys are read from files during startup of the application handling the TLS/SSH connection and stay in memory until the application terminates.

All ephemeral keys are held by OpenSSL in cipher handles which contains the memory buffer holding the keys. When the cipher handle is released, the key memory is zeroized.

The following keys are managed by the Linux kernel crypto API on behalf the disk ecryption mechanism of dm-crypt:

- Ephemeral symmetric key: The master volume key is stored wrapped in the LUKS header. That header is read by the tool cryptsetup that uses libgcrypt as its cryptographic library. The LUKS header is unwrapped using a passcode provided by the administrator. The unwrapped master volume key is injected into the kernel crypto API which wraps the key in a cipher handle that contains the memory buffer holding the key. When the dm-crypt partition is unmounted, the Linux kernel crypto API cipher handle is deallocated which also zeroizes the memory buffer holding the master volume key. After cryptsetup injected the master volume key, the passcode, the key derived from it using PBKDF2 as well as its copy of the master volume key is overwritten with zeros.

For non-volatile memory, the life of a TLS certificate / key and SSH key pairs is indefinite. A user or administrator can manually destroy them. To erase long-term key material held in files the TOE provides the tool fstrim. After a deletion of a file with sensitive data, this tool uses the SSD TRIM command to inform the SSD to discard unused blocks bypassing wear leveling. In addition, the tool shred is available that overwrites files multiple times with random data. This tool can be used for HDD to delete data.

As the LUKS header only holds the wrapped master volume key, it is not subject to zeroization requirements.

This supports FCS_CKM_EXT.4.

7.1.9 Cryptographic operation - Encryption/Decryption (Disk Encryption)

AES-XTS is used for the disk encryption based on dm-crypt. The cipher is implemented within the Linux kernel crypto API.

AES-CBC and AES-GCM are used for TLS as well as SSH bulk data encryption.

For details, see Cryptographic services.

This supports FCS_COP.1(1).

7.1.10 Cryptographic operation - Hashing

The hashing operation is applied for the following mechanisms:

- Signature generation / verification: The TOE performs signature generation using SHA2. Signature verification is supported with SHA1 and SHA2.
- MAC: The hash implementation is used as part of HMAC.

The technical decision 578 explicitly discourages signatures with SHA-1 as provided with ssh-rsa. This ST includes this signature type for legacy support only and strongly recommends using one of the other mentioned signature types.

For details, see Cryptographic services.

This supports FCS_COP.1(2).

7.1.11 Cryptographic operation - Signing

For user and host authentication, RSA and ECDSA signature generation and verification are used as part of the SSH protocol.

To support server authentication as part of TLS, RSA and ECDSA signature generation and verification are used as part of the TLS protocol.

For details, see Cryptographic services.

This supports FCS_COP.1(3).

7.1.12 Cryptographic operation - Keyed-Hash Message Authentication

Keyed hash message authentication is applied for the following operations:

- Authenticated encryption as part of SSH / TLS
- PRF: The SSH / TLS PRF requires HMAC.
- PBKDF2: The key derivation from the user passcode during unwrapping of the LUKS header requires HMAC.

For details, see Cryptographic services.

This supports FCS_COP.1(4).

7.1.13 Random Bit Generation

Random bit generation is performed in the following cryptographic libraries:

- OpenSSL: The CTR AES DRBG with AES 256 core with DF, without PR is used to generate all key material for the SSH/TLS protocols.
- libgcrypt: The HMAC DRBG with SHA-256 core is used to generate the master volume key when initializing a dm-crypt disk encryption container.

For details, see Cryptographic services.

This supports FCS_RBG_EXT.1.

7.1.14 Storage of Sensitive Data

The TOE provides disk encryption for partitions. All partition including the root partition can be encrypted. In case of encryption of the root partition, the kernel and the initial RAM disk files must be stored on an unencrypted partition. These two files together with the boot loader configuration file cannot be stored protected by dm-crypt.

For details, see Cryptographic services.

This supports FCS_STO_EXT.1.

7.1.15 TLS Client Protocol - Part 1

The TLS client support is implemented in OpenSSL. It supports all cipher suites listed in FCS_TLSC_EXT.1.

The authentication of the TLS server certificate is performed as follows:

- Using FQDN: When the server's FQDN can be resolved, the TOE tries to match its name with either the CN part of the DN or with the DNS Name or URI Name entry in the SAN. If a CN and a SAN are present, the SAN takes precedence.

- Using IP address: When the server's FQDN cannot be resolved, the TOE tries to match its IP address with either the CN part of the DN or with the IP entry in the SAN. If a CN and a SAN are present, the SAN takes precedence.

The TOE supports wild cards for the server identifier resolution.

Certificate pinning is not supported by the TOE.

For details, see Cryptographic services.

This supports FCS_TLSC_EXT.1.

7.1.16 TLS Client Protocol - Part 2

The TOE supports the TLS supported groups extension in the client hello without specific configurations.

For details, see Cryptographic services.

This supports FCS_TLSC_EXT.2.

7.1.17 Access Controls for Protecting User Data

The TOE access control functionality is described in Discretionary Access Control.

This supports FDP_ACF_EXT.1.

7.1.18 Management of security functions behavior

The TOE management functionality is described in Security Management.

This supports FMT_MOF_EXT.1.

7.1.19 Specification of Management Functions

The TOE management functionality is described in section Security Management.

This supports FMT_SMF_EXT.1.

7.1.20 Access controls

The TOE files are located in the following directories:

- Kernel / initial RAM disk / boot loader configuration: /boot
- Kernel modules: /lib/modules
- Shared libraries: /lib, /lib64, /usr/lib, /usr/lib64
- System executables: /bin, /sbin, /usr/bin, /usr/sbin, /usr/libexec
- System configuration data: /etc
- Security audit logs: /var/log/audit

This supports FPT_ACF_EXT.1.

7.1.21 Address Space Layout Randomization

ASLR is applied to all ELF formatted user space executables and its libraries.

This supports FPT_ASLR_EXT.1.

7.1.22 Stack Buffer Overflow Protection

The TOE implements stack canaries for each executable and its libraries. The stack canary is implemented as a random number that is located between the stack and the return address pointer. The compiler adds a check for the presence of an unchanged random number before a function completes and before the function return pointer is executed.

This supports FPT_SBOP_EXT.1.

7.1.23 Boot Integrity

When a computer with Linux is turned on, the operating system is loaded into memory by a special program called a boot loader. A boot loader usually exists on the system's primary hard drive, or other media device, and has the sole responsibility of loading the Linux kernel with its required files or, in some cases, other operating systems, into memory. Each architecture capable of running Linux uses a different boot loader.

The init process provided with the systemd suite and is the ancestor of all userspace processes and is process ID 1. Its main functions are to start processes and daemons based on the contents of the /etc/systemd/system, /usr/lib/systemd/ and /lib/systemd directories, to reap child processes, and to react to certain signals, such as start or stop signals.

7.1.23.1 Boot loader operation

A boot loader is a program that resides in the starting sectors of a disk, that is, the Master Boot Record (MBR) of the hard disk. After testing the system during boot, the Basic Input-Output System (BIOS) transfers control to the MBR if the system is set to be booted from there. Then the program residing in MBR gets executed. This program is called the boot loader. Its duty is to transfer control to the operating system, which will then proceed with the boot process.

The boot process consists of the following steps when the CPU is powered on or reset:

1. The firmware performs any hardware initialization steps.
2. The BIOS searches for the boot loader to boot in an order predefined by the firmware setting. Once a valid device is found, the firmware copies the contents of its first sector containing the boot loader into RAM, and starts executing the code just copied.

Every boot loader performs the following general steps to initialize Linux:

1. Loading the kernel image it is configured to load (the actual way of configuring the boot loader is different for each boot loader implementation). The loading process ensures that the kernel image is loaded to a well-defined memory location.
2. Loading the initramfs image it is configured to load. Again, this image is loaded to a well-defined memory location.
3. The kernel is compiled such that the setup function will always be loaded into a well-known memory location. This allows the boot loader to jump to the setup function to transfer control to the kernel.

The system firmware supports the boot integrity as follows:

- x86: UEFI implements the starting of the boot loader. During the start, UEFI performs a signature verification of the first stage boot loader executable called "shim boot loader" that also holds the certificate for validating the Grub boot loader. That signature is part of the boot loader EFI file and is signed by Microsoft since the default UEFI certificate in every system is the Microsoft certificate. The purpose of the "shim boot loader" is to perform the integrity check of the Grub boot loader binary and the certificate used to verify the subsequent software images. In turn, the Grub boot loader verifies the integrity of the kernel binary file. The loaded binary has a PKCS #7 signature at the end of the file which is used by the secure boot process to perform signature validation.

- IBM Z: The loaded binaries have a signature associated with the file which is used by the secure boot process to perform signature validation.
- ARM: The loaded binaries have a signature associated with the file which is used by the secure boot process to perform signature validation.

7.1.23.2 Kernel boot process

The following initialization process is followed by the kernel. The details of the boot process are very different for each architecture. However, the following high-level steps are followed by each architecture.

Note, the kernel binary is compressed, except for a small code portion. That portion contains the setup code and the decompression routines in the kernel code to allow the kernel code to decompress itself.

The following steps are performed by the kernel after being loaded by the boot loader.

1. The setup function reinitializes the hardware devices in the computer and sets up the environment for the execution of the kernel program. The setup function initializes and configures hardware devices, such as the keyboard, video card, disk controller, and floating point unit.
2. The kernel is loaded into memory, and if its a compressed image, it is decompressed.
3. The kernel calls a second start-up (e.g. startup_32 on x86) function to set the execution environment for process 0.
4. The kernel initializes the memory management system.
5. The kernel sets the kernel mode stack for process 0.
6. The kernel initializes the provisional Page Tables and enables paging.
7. The kernel sets up the exception handlers.
8. start_kernel completes the kernel initialization by initializing Page Tables, Memory Handling Data Structures, the SLUB allocator, system date, and system time.

7.1.23.3 User space boot process

After the kernel is fully initialized, the user space is started up. There are the following two phases covering the boot process:

- initramfs: This state is intended to perform any initialization work to make the root file system available, such as loading kernel modules with special drivers needed to access the non-volatile storage holding the root file system.
- systemd This state initializes the entire user space by loading applications and daemons and performs any setup and configuration process necessary to get the system into the operational state.

initramfs

The following steps are performed to initialize the initramfs:

1. After the kernel is loaded and initialized, it locates the compressed initramfs image in memory.
2. The Linux kernel uncompresses the image.
3. The kernel performs a loopback mount of the uncompressed initramfs image to mount it as the root file system.
4. The kernel executes the /linuxrc or /sbin/init executable. This is a copy of systemd which executes out of the initramfs.
5. systemd does whatever it needs to do to for setting up the system to allow accessing the root file system based on the configuration.

6. After the systemd application terminates, the kernel unmounts the initramfs, and mounts the root file system pointed to by the “root” kernel command line parameter.

systemd

On every Unix system there is one process with the special process identifier 1. It is started by the kernel before all other processes and is the parent process for all those other processes that have nobody else to be child of. Due to that it can do a lot of stuff that other processes cannot do. And it is also responsible for some things that other processes are not responsible for, such as bringing up and maintaining userspace during boot.

systemd starts up and supervises the entire system (hence the name...). It is based around the notion of units. In systemd, a unit refers to a resource that is managed. Each resource is defined by a configuration file called a unit file. Example: a unit avahi.service is the unit file for the Avahi daemon. Units are categorized by the type of their resource. The suffix portion of the unit's file name is the type.

The generic boot sequence after the initramfs operation is finished is given in the following. The kernel has mounted the root file system which hosts /sbin/init – note that this init application is implemented with the systemd framework. This application is the driver of the user space boot process.

The kernel executes /sbin/init which finalizes the boot sequence implemented in the kernel.

1. Before executing the /sbin/init application, it resets the pid table to assign process ID one to the init process.
2. systemd is an event-driven system as described in systemd(8).
3. The boot process driven by systemd is purely based on events. If one event is observed, all tasks associated with that event are executed in parallel. The implemented boot sequence with all events is outlined in bootup(7). The boot process covers the following aspects:
    a. Mounts the /proc special file system.
    b. Mounts the /dev/pts special file system.
    c. Generate the /etc/nologin file early in the boot process.
    d. Saves and restores the system entropy tool for higher quality random number generation.
    e. Configures network interfaces.
    f. Starts the system logging daemons.
    g. Starts the sshd daemon.
    h. Starts the cron daemon.
    i. Probes hardware for setup and configuration.
    j. Enables the logind daemon for authentication.

This supports FPT_TST_EXT.1.

7.1.24 Trusted Update

The system software management provided by the zypper tool obtains a list of updates regularly from SUSE. The list is compared with the installed set of RPMs. If an update for an installed RPM is found, the administrator is notified about the presence of the update.

During installation, the TOE receives a certificate that is stored in with the RPM database. That certificate is used to validate every RPM that is installed. A missing signature or wrong signature of an RPM package causes the package to be not installed. The TOE and its updates are distributed as RPMs.

This supports FPT_TUD_EXT.1.

7.1.25 Trusted Update for Application Software

The functionality for the system update discussed for FPT_TUD_EXT.1 applies to application software as well.

This supports FPT_TUD_EXT.2.

7.1.26 Audit data generation

The TOE audit generation mechanism is described in Audit.

This supports FAU_GEN.1.

7.1.27 Authentication failure handling

The TOE identification and authentication mechanism is described in Identification and Authentication.

This supports FIA_AFL.1.

7.1.28 Multiple Authentication Mechanisms

The TOE identification and authentication mechanism is described in Identification and Authentication.

This supports FIA_UAU.5.

7.1.29 X.509 Certificate Validation

The TOE X.509 certificate validation mechanism enforces the following checks:

- Firstly a certificate chain is built up starting from the supplied certificate and ending in the root CA. It is an error if the whole chain cannot be built up. The chain is built up by looking up the issuers certificate of the current certificate. If a certificate is found which is its own issuer it is assumed to be the root CA.
- The process of 'looking up the issuers certificate' itself involves a number of steps. After all certificates whose subject name matches the issuer name of the current certificate are subject to further tests. The relevant authority key identifier components of the current certificate (if present) must match the subject key identifier (if present) and issuer and serial number of the candidate issuer, in addition the keyUsage extension of the candidate issuer (if present) must permit certificate signing.
- The lookup first looks in the list of untrusted certificates and if no match is found the remaining lookups are from the trusted certificates. The root CA is always looked up in the trusted certificate list: if the certificate to verify is a root certificate then an exact match must be found in the trusted list.
- The second operation is to check every untrusted certificate's extensions for consistency with the supplied purpose. The supplied or "leaf" certificate must have extensions compatible with the supplied purpose and all other certificates must also be valid CA certificates. The following checks are applied:
    - The basicConstraints extension CA flag is used to determine whether the certificate can be used as a CA. If the CA flag is true then it is a CA, if the CA flag is false then it is not a CA. All CAs should have the CA flag set to true.

- If the basicConstraints extension is absent then the certificate is considered to be a "possible CA" other extensions are checked according to the intended use of the certificate. A warning is given in this case because the certificate should really not be regarded as a CA: however it is allowed to be a CA to work around some broken software.
- If the certificate is a V1 certificate (and thus has no extensions) and it is self signed it is also assumed to be a CA but a warning is again given: this is to work around the problem of Verisign roots which are V1 self signed certificates.
- If the keyUsage extension is present then additional restraints are made on the uses of the certificate. A CA certificate must have the keyCertSign bit set if the keyUsage extension is present.
- The extended key usage extension places additional restrictions on the certificate uses. If this extension is present (whether critical or not) the key can only be used for the purposes specified. The following purposes are defined:
    - SSL Client: The extended key usage extension must be absent or include the "web client authentication" OID. keyUsage must be absent or it must have the digitalSignature bit set. Netscape certificate type must be absent or it must have the SSL client bit set.
    - SSL Client CA: The extended key usage extension must be absent or include the "web client authentication" OID. Netscape certificate type must be absent or it must have the SSL CA bit set: this is used as a work around if the basicConstraints extension is absent.
    - SSL Server: The extended key usage extension must be absent or include the "web server authentication" and/or one of the SGC OIDs. keyUsage must be absent or it must have the digitalSignature, the keyEncipherment set or both bits set. Netscape certificate type must be absent or have the SSL server bit set.
    - SSL Server CA: The extended key usage extension must be absent or include the "web server authentication" and/or one of the SGC OIDs. Netscape certificate type must be absent or the SSL CA bit must be set: this is used as a work around if the basicConstraints extension is absent.
    - Netscape SSL Server: For Netscape SSL clients to connect to an SSL server it must have the keyEncipherment bit set if the keyUsage extension is present. This isn't always valid because some cipher suites use the key for digital signing. Otherwise it is the same as a normal SSL server.
    - Common S/MIME Client Tests: The extended key usage extension must be absent or include the "email protection" OID. Netscape certificate type must be absent or should have the S/MIME bit set. If the S/MIME bit is not set in Netscape certificate type then the SSL client bit is tolerated as an alternative but a warning is shown: this is because some Verisign certificates don't set the S/MIME bit,
    - S/MIME Signing: In addition to the common S/MIME client tests the digitalSignature bit or the nonRepudiation bit must be set if the keyUsage extension is present.
    - S/MIME Encryption: In addition to the common S/MIME tests the keyEncipherment bit must be set if the keyUsage extension is present.
    - S/MIME CA: The extended key usage extension must be absent or include the "email protection" OID. Netscape certificate type must be absent or must have the S/MIME CA bit set: this is used as a work around if the basicConstraints extension is absent.

    - CRL Signing: The keyUsage extension must be absent or it must have the CRL signing bit set.
    - CRL Signing CA The normal CA tests apply. Except in this case the basicConstraints extension must be present.

- The third operation is to check the trust settings on the root CA. The root CA should be trusted for the supplied purpose. For compatibility with previous versions of OpenSSL, a certificate with no trust settings is considered to be valid for all purposes.
- The final operation is to check the validity of the certificate chain. The validity period is checked against the current system time and the notBefore and notAfter dates in the certificate. The certificate signatures are also checked at this point.

If all operations complete successfully then certificate is considered valid. If any operation fails then the certificate is not valid.

This supports FIA_X509_EXT.1 and FIA_X509_EXT.2.

7.1.30 Trusted channel communication

The TOE trusted channel support is described in Cryptographic services.

This supports FTP_ITC_EXT.1.

7.1.31 Trusted Path

The TOE is administered remotely by using SSH to access the TOE.

This supports FTP_TRP.1.

7.1.32 Cryptographic operation - Encryption/Decryption (SSH)

The TOE allows the use of AES-CTR with SSH. The counter value is derived from the shared secret obtained during the SSH handshake like the symmetric key. The counter is incremented by one and is a 32 bit value. Due to the maximum life time of a key, the counter value can never overflow.

This supports FCS_COP.1(5).

7.1.33 SSH Protocol

The TOE supports the following authentication mechanisms with SSH:

- Password-based authentication
- Key-based authentication with RSA keys and ECDSA keys.

The TOE maintains a counter for each SSH packet which is increased by the number of received bytes. If the counter reaches the threshold of 262144 bytes, the connection is closed. After processing at most 2^28 packets, the TOE initiates a re-keying.

The SSH implementation supports the following ciphers:

- Symmetric ciphers: aes128-ctr, aes256-ctr, aes128-cbc, aes256-cbc, AES128 GCM, AES256 GCM
- Asymmetric ciphers: ssh-rsa, rsa-sha2-256, rsa-sha2-512, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384
- MAC: hmac-sha1, hmac-sha2-256, hmac-sha2-512, AES GCM
- Key agreement: diffie-hellman-group14-sha1, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521

The technical decision 578 explicitly discourages signatures with SHA-1 as provided with ssh-rsa. This ST includes this signature type for legacy support only and strongly recommends using one of the other mentioned signature types.

The aforementioned statements apply equally to the SSH client and server implementation.

For details, see Cryptographic services.

This supports FCS_SSH_EXT.1, FCS_SSHC_EXT.1, FCS_SSHS_EXT.1.

7.1.34 Timely Updates

The entire TOE (and in fact the entire SLES distribution) is subject to an extensive update process. The update process starts when SUSE is informed about defects. Depending on the severity (security incidents are considered to be severe), fixes are developed, tested and released with updated RPM packages.

The entire update process is handled by SUSE and covers all packages shipped as part of the SLES distribution from which the TOE is a subset.

Guaranteed response times depend on the selected service level agreement which is outlined in https://www.suse.com/support/handbook/. The security incident and response process allows customers to directly interact with the SUSE team via a central email address and PGP key provided at https://www.suse.com/support/security/contact/ to report issues. Based on a timely triage and root cause analysis a responsible resolution of the incident report is ensured which may result in the release of an update of the affected software binaries. Such updates are made available via the automated update channels to all customers.

Identified issues can be relayed to SUSE either via the support channels defined by the service level agreement or via the communication specified in https://www.suse.com/support/security/.

This supports ALC_TSU_EXT.1.

8 Abbreviations, Terminology and References

8.1 Abbreviations

ACL
Access Control List

API
Application Programming Interface

KVM
Kernel Virtualized Machine

HTTP
Hypertext Transfer Protocol

SFR
Security Functional Requirement

SSL
Secure Sockets Layer

ST
Security Target

TCP/IP
Transmission Control Protocol / Internet Protocol

TLS
Transport Layer Security

TOE
Target of Evaluation

TSF
TOE Security Functionality

VM
Virtual Machine

VPN
Virtual Private Network

8.2 Terminology

This section contains definitions of technical terms that are used with a meaning specific to this document. Terms defined in the [CC] are not reiterated here, unless stated otherwise.

AppArmor
Linux kernel LSM module that is able to implement additional restrictions for executables. This LSM is unused in the evaluated configuration.

Authentication Data
Authentication data is the data used by users or remote entities to authenticate their claimed identity.

Authorized Administrator
This term refers to a user in one of the defined administrative roles of a Linux system. The TOE associates the user with the UID of zero and named "root" with administrative authorities. Effectively, the UID zero is assigned with all Linux capabilities known to the Linux kernel. Every user who is allowed to log on as that root user, or to switch their UID to the root user is considered an authorized administrator. In addition, any user who is

able to execute applications which grant one or more Linux capabilities to be used in an unconditional manner is considered an authorized administrator. Note: the process executing on behalf of the root user must possess MLS override attributes to perform management aspects of the Authoritative Access Control Policy.

Category
A category is the non-hierarchical category of the lower MLS label defined with an SELinux label. Note: an SELinux label consists of four parts where the MLS label is one of them. The MLS label in turn is split into a higher and a lower MLS label part.

Classification
A sensitivity label associated with an object.

Clearance
A sensitivity label associated with a subject or user.

DAC
Discretionary Access Control implemented with permission bits and ACLs.

Data
Arbitrary bit sequences on persistent or transient storage media.

Guest
Software executing within a virtual machine environment. There can be zero or more guests executing concurrently on the host system.

Host
The host system provides the Linux environment that controls and manages the virtual machines. The host provides the execution environment for every virtual machine.

Information
Any data held within a server, including data in transit between systems.

IOMMU
Input / Output Memory Management Unit. This MMU allows the setup of multiple DMA areas for different virtual machines.

KVM
Kernel-based Virtual Machine.

MLS
Multi-level security

Named Object
In Linux, those objects that are covered by access control policies. The list of objects defined as named objects is provided with SPM

Object
For Linux, objects are defined by SPM.

OSPP
Protection Profile for General Purpose Operating Systems

PAM
Pluggable Authentication Module - the authentication functionality provided with Linux is highly configurable by selecting and combining different modules implementing different aspects of the authentication process.

Product
The term product is used to define software components that comprise the SLES system.

QEMU
The QEMU software component implements the virtual devices and virtual resources for virtual machines. There is one instance of QEMU per virtual machine. The QEMU software component is also identified as the "kvm" application on the host system.

SELinux
Linux kernel LSM module that is able to implement arbitrary security policies. An SELinux policy distributed with the TOE implements multi-level or multi-category security.

Subject
There are two classes of subjects in the TOE: i) untrusted internal subject - this is a Linux process running on behalf of some user or providing an arbitrary service, running outside of the TSF (for example, with no privileges); ii) trusted internal subject - this is a Linux process running as part of the TSF (for example: service daemons and the process implementing the identification and authentication of users).

Target Of Evaluation (TOE)
The TOE is defined as the SUSE Linux Enterprise Server operating system, running and tested on the hardware and firmware specified in this Security Target. The BIOS firmware as well as the hardware are not part of the TOE.

User
Any individual/person or technical entity (such as a service added by the administrator on top of the TOE) who has a unique user identifier and who interacts with the SLES product.

User Security Attributes
Every user is associated with a number of security attributes which allow the TOE to enforce its security functions on this user. This also includes the user clearance which defines the maximum sensitivity label a user can have access to.

Virtual devices
See virtual resources for a generic explanation. This definition applies also to virtual devices, but with a focus to devices, such as disks, network cards, graphics cards, and similar.

Virtual machine
A virtual machine is an execution environment where the software executing within the virtual machine has access to the processor's user and supervisor state and resources defined by the host system. Resources include the number of processors, RAM size, physical devices, virtualized devices, communication channels to other virtual machines and the host system. For the KVM environment a virtual machine environment is controlled and provided by the Linux kernel hypervisor functionality plus the QEMU application instantiated for each virtual machine.

Virtual machine environment
See virtual machine.

Virtual resources
Virtual resources are resources that either do not physically exist and do not exist in the host system. Virtual resources are implemented by the virtual machine environment and are provided to the respective virtual machine. For example, virtual resources are special exceptions that can be triggered from the virtual machine environment to request services from the host system, such as para-virtualized drivers. Virtual devices can be considered one form of virtual resources.

8.3 References

CC	**Common Criteria for Information Technology Security Evaluation**	
	Version	3.1R5
	Date	April 2017
	Location	http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf
	Location	http://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R5.pdf
	Location	http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf
OSPP	**Protection Profile for General Purpose Operating Systems**	
	Version	4.2.1
	Date	2019-04-22
RFC4252	**The Secure Shell (SSH) Authentication Protocol**	
	Date	January 2006
	Location	http://tools.ietf.org/html/rfc4252
RFC4253	**The Secure Shell (SSH) Transport Layer Protocol**	
	Date	January 2006
	Location	http://tools.ietf.org/html/rfc4253
SSHEP	**Extended Package for Secure Shell (SSH)**	
	Version	1.0
	Date	2016-02-19