MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Hardware-encrypted disks in Linux

Master's Thesis

ŠTĚPÁN HORÁČEK

Brno, Fall 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Hardware-encrypted disks in Linux

Master's Thesis

ŠTĚPÁN HORÁČEK

Advisor: Ing. Milan Brož, Ph.D.

Department of Computer Systems and Communications

Brno, Fall 2022


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Štěpán Horáček

Advisor: Ing. Milan Brož, Ph.D.

Abstract

The thesis aims to analyze existing approaches to using hardware-encrypted block devices (disks) in Linux.

The implementation part should provide basic low-level tools for tools configuration and status check of such devices. Tools should use generic Linux kernel interfaces (like ioctl calls).

Student should

- get familiar with and study available resources for self-encrypted drives, OPAL2 standard, block layer inline encryption,
- analyze and describe security of such drives,
- provide state-of-the-art overview of existing attacks,
- implement proof-of-concept low-level tools to access available devices,
- evaluate and discuss the result.

The student should be familiar with C code for low-level system programming and cryptography concepts.

Keywords

keyword1, keyword2, ...

Contents

1 Introduction

Whether it is a disk of a corporate laptop, personal desktop, or server storage, all of these devices may very likely contain sensitive information such as company internal data, medical records, or even session cookies. According to Verizon 2022 Data Breach Investigation Report [1], lost or stolen assets made up almost 5% of the analysed security incidents (not necessarily data breaches due to the way it's hard to find out if there was one with lost or stolen assets). Furthermore, this percentage does not account for other possible attacks, where the disk could be compromised without stealing the device, such as unauthorized access to the deployed server.

other than merketing talks, nothing concrete about advantage....

As such, it can be seen that some form of protection for the confidentiality of data on the disk is necessary. Disk encryption may seem like an obvious solution. However, traditional software disk encryption has several flaws. Such as increased energy demand or requirement of keys being present in the memory at all times, presenting a vulnerability. Even though the transfer of responsibility for encryption solves these issues, it introduces other new ones.

In this thesis, we will introduce the idea of hardware disk encryption, the basic terms used in this area, and the categorisation of different approaches to hardware disk encryption. Afterwards, we will focus on the Opal standard as a representative of the self-encrypting disks approach to hardware disk encryption. Next, we will look at possible attacks on devices using hardware disk encryption. Finally, we will introduce our own tool to manage Opal disks and get their properties, and analyse data acquired using this tool.

2 Hardware disk encryption

Hardware disk encryption is a technology that provides confidentiality of data stored on a storage device using encryption provided by hardware specialized for encrypting data transferred to and decrypting data transferred from disk.

Generally, disk encryption is done by encrypting the saved data using a data encryption key (DEK), also sometimes known as the media encryption key. In order to allow changing of passwords and having multiple passwords, the DEK is not cryptographically bound to any particular password and instead is saved in an encrypted form. It is encrypted using a key encryption key (KEK), which is generated from a password and therefore is cryptographically bound to a specific password.

The data encryption process can be conducted in logically and physically different places, depending on the type of disk encryption. In the following sections of this chapter, we will further describe three such types.

2.1 Self-encrypting drives

Self-encrypting drive (SED) is a storage device with built-in disk encryption hardware. Having built-in disk encryption hardware enables the disk to automatically encrypt the incoming data and decrypt the outgoing data. The data saved on the disk is protected two-fold: by access control and by encryption. Access control denies any input-output operation until the host is authorized. Encryption protects the device even from attacks with physical access, where the access control is bypassed.

The encryption of the data is usually active even if the security mechanism of the device are not activated and the device is not protected by a password. This has the advantage that a non-activated disk can be activated and protected by a password without a need to reencrypt all data on the disk SEDs also might not require management by the host system after the initial unlocking. Even if multiple sections with different keys or passwords are defined on the disk, after they are unlocked no future key management is required. Thanks to the spe-

cialized encryption chip, SEDs have not only increased performance, reduced CPU usage, and reduced power consumption [2]. Since the encryption keys are not actively used by the host system, they do not have to be in the memory, which provides protection against malicious software running in the host system.

In order to provide an interface to control the encryption of the disk, such as unlocking or changing the password, there exist several standards. Other than the series of standards defined by the Trusted Computing Group discussed later in chapter 3, there are other standards used by SEDs.

2.1.1 ATA Security Feature Set

ATA Security Feature Set [3] allows one to protect the data and configuration of the disk. It offers commands to set a password, unlock the disk, erase the data on the disk, freeze the device, and disable the password.

In order to authenticate to the disk, there are two passwords, the Master password, initially set by the manufacturer, and the User password. The usage of the Master password to unlock or disable the security can be disabled by setting the Master Password Capability bit while setting the User password. Even if the Master password is disabled for unlocking the disk or disabling the locking, it can still be used for erasing the disk. The disk protection is enabled by setting the User password. After the device is turned on again, the device is locked and to access the data on the disk or use some of the ATA commands, and it needs to be unlocked by providing a password using the unlock command. By disabling the User password, the protection of the device can be disabled.

However, the ATA Security Feature Set itself does not provide encryption of the data saved on the disk, only access control. Nevertheless, some of the SED solutions use ATA Security commands as their interface to provide access to their own implementation of the disk encryption and disk unlocking [4].

2.1.2 Proprietary solutions

There also exist proprietary alternatives to the previously introduced standards. Such disks are then often managed either by software downloaded from the manufacturer's site or by using software from a special partition on the disk. An example of such are disks from Western Digital's My Passport and My Book series [5], or disks in IBM PureData System for Analytics N3001 [6].

is it okay to cite vulnerability paper in such case? maybe find something else to widen the horizons.

2.2 Inline encryption hardware

Compared to the previously described self-encrypting drives, inline encryption hardware is separated and independent from the disk. Inline encryption hardware is similar to classic crypto accelerators. However, inline encryption hardware, instead of using the same memory for input and output, optimizes the input-output process by writing and reading the data directly to and from the disk.

Fact check this. Maybe just integrate the subsection into something else.

Inline encryption hardware shares several of SEDs advantages, such as reduced power consumption, reduced CPU usage, or increased performance, compared to purely software encryption. However, unlike SEDs, inline encryption devices usually do not have any access control or authentication capabilities. Instead, they allow one to simply insert a DEK into a key slot, select a key slot, and remove a DEK from a key slot. The data passing through the inline encryption device to the disk is then encrypted using the DEK from the selected key slot. Similarly, data passing from the disk are decrypted by the inline encryption hardware. Since key management is entirely in control of the host system, which has knowledge of the file system mapping, inline encryption can be used not only for block-level encryption but also for filesystem-level encryption.

An example of inline encryption hardware standard is the Qualcomm Inline Crypto Engine [7] used by some Android devices.

some nist document or maybe some android kernel source

2.3 Software disk encryption

As an alternative to hardware encryption, there exists software disk encryption. Compared to hardware disk encryption, software disk

something about how it provides actually a way to check the encrypted content, right? Seems to be primarily in android , source? Find how much inline enc hardware there re-

encryption does not require specialized hardware. However, even though specialized hardware is not required, performance is often improved using cryptographic coprocessors.

probably just a quick overview, ... maybe change the chapter to just "disk encryption"... or mention this just shortly at the start...

2.3.1 Linux stack

There are several available software encryption libraries and subsystems in the Linux kernel. In this section, we will mention only two of them, each representing a different approach to software encryption.

dm-crypt The subsystem *dm-crypt* works on the block device level. It can encrypt any data, including metadata or partition tables, but it does not offer precise control over what is encrypted, and the entire filesystem must be encrypted.

In order to control dm-crypt encryption, there exists a tool called cryptsetup, described in more detail later in chapter 4.

rethink where to put this... maybe tools?

fscrypt The kernel space library *fscrypt* works on a filesystem level. It is more flexible compared to the block device level dm-crypt, as it can limit the encryption to only some files. However, fscrypt does not encrypt all information for that file, and the metadata (excluding the filename) are left unencrypted.

Although fscrypt offers software encryption by default, inline encryption is also supported. The usage of inline encryption in fscrypt is described in more detail later in chapter 4.

There also exists a user space application with the same name[1], that can be used to control encryption of file systems using kernel space fscrypt.

I don't think that inline encryption is supported AFAIK, does not require any extra implementations on the side of drivers or file system or anything, right?

1. `https://github.com/google/fscrypt`

3 TCG Opal 2.0

TCG Opal Security Subsystem Class (SSC) 2.0 (hereinafter referred to simply as "Opal standard") is a specification for storage devices, aiming to provide confidentiality of stored data while the conforming disk is powered off [8]. It is one of the representatives of the self-encrypting drive approach to hardware disk encryption. Information for this chapter comes primarily from the Core standard [9] and the Opal standard [8].

In the following sections, we shall first describe the specification, its features and capabilities as described by the Core and Opal standards. Afterwards, we will focus on the mandatory features and requirements on Opal devices, and lastly, we will describe the interface available on a Linux host.

maybe split it into TCG storage and TCG opal chapters/-sometghings.

maybe move this out of the chapter into a separate chapter,,, with external tools... yea

3.1 Structure of the standard

The Opal standard is defined as a subsystem extending the TCG Storage Architecture Core Specification (hereinafter referred to simply as "Core standard"). The Core standard [9] specifies the core features and properties shared among several different types of storage security subsystem classes (SSC), that extend the core functionality by specifying additional features or define the set of mandatory features. Each of these subsystems is focused on a different use case. These subsystems are namely:

- Opal — targeted at a corporate and personal usage. As this is the most available subsystem, it will be the target of our focus and be described more closely in the rest of this chapter.
- Opalite — simplified Opal. Does not mandate features such as locking ranges, decreases the minimal number of admin and user authorities, or additional DataStore tables [10].
- Pyrite — encryption-less Opalite. Similar to Opalite, however it does not mandate encryption of data saved on disk, and instead may offer only logical access control [11].

- Ruby — focused on data centers and server drives. Offers only global range encryption, weaker configuration of access control, no pre-boot authentication support [12]. Replacing the older Enterprise subsystem.

Other than the Core and SSC standards defining the fundamentals of the devices, there are defined also Feature Sets. These Feature Sets expand the standards with less fundamental features. Some of those are the Additional DataStore [13], providing a storage for arbitrary data, the Block SID Authentication [14], allowing to disable the owner's authority until a power cycle, the PSK Secure Messaging [15], providing pre-shared key secure communication between the host and the device, the PSID [16], defining a way to reset the device into the manufactured state, or the Single User Mode [17]. However, we will focus primarily on those that are mandatory in the Opal specification.

3.2 Architecture

The Core standard [9] splits the storage device implementing it into two parts: the trusted peripheral and security providers.

maybe just mix into another section.

A trusted peripheral (TPer) is a component located on the disk that provides the security of the data on the disk. TPer mediates the communication between its Security Providers and the host.

Security Provider (SP) is defined as a set of tables, methods and access control. SPs are derived from a set of templates, which define a set of tables and methods aimed at one functionality, a subset of which the SP implements. The templates are described more closely in a later section 3.7. The Opal 2.0 standard defines that at least the Admin SP and Locking SP must be present in the TPer. The Admin SP is tasked with administrating the TPer and other SPs, which may include creating new SPs, deleting existing SPs, or providing information about SPs. The Locking SP provides access to functionality such as managing locking ranges, locking the drive, or managing access control. Both of these SPs are described more closely in later sections 3.8 and 3.9.

3.3 Capability discovery

In order to find out the properties and features of a particular device, there exists the so-called Discovery process. This process is divided into three levels, each with different reported information and a different approach to accessing the information.

3.3.1 Level 0 Discovery

Level 0 Discovery provides basic information about the secure device and is performed using special IF-RECV command of the device. The information about the device is provided through feature descriptors. The presence of a feature descriptor header means that the feature is supported, and the fields of the header describe the basic properties of that feature.

introduce IF-RECV earlier?

Some of the frequent features described by these descriptors are:

- TPer Feature Descriptor — reports supported communication features such as ACK/NACK support, ComID management, buffer management, or asynchronous communication.
- Locking Feature Descriptor — reports information relevant to the Locking SP, such as support of data encryption, whether the disk is locked or the status of master boot record shadowing.
- Geometry Feature Descriptor — reports information about the geometry of the disk, such as block size or alignment granularity.
- SingleUser Feature Descriptor — reports information about SingleUser Feature Set, such as whether it is activated,
- DataStore Feature Descriptor — reports information about DataStore Feature Set, such as the maximum size of the DataStore table or maximum count of DataStore tables.
- Opal 2.0 Feature Descriptor — reports information specific to Opal SSC, such as the base ComID, number of ComIDs, or the maximum number of users and admins authorities of Locking SP.

3.3.2 Level 1 Discovery

Level 1 Discovery provides information about the capabilities of the communication channel and is performed using the `Properties` control session method. The Level 1 Discovery is used not only as a way for the host to find out the capabilities of the TPer, but to determine shared limits depending on the capabilities of both the TPer and the host, as the host can also communicate its capabilities to the TPer.

introduce control session method earlier?

The reported properties mandatory for Opal are:

- `MaxMethods` — maximum number of methods per received SubPacket.
- `MaxSubpackets` — maximum number of SubPacket per received Packet.
- `MaxPacketSize` — maximum size of a received Packet.
- `MaxPackets` — maximum number of Packets per received ComPacket.
- `MaxComPacketSize` — maximum size of a received ComPacket.
- `MaxResponseComPacketSize` — maximum size of sent ComPacket.
- `MaxSessions` — maximum number of active sessions.
- `MaxIndTokenSize` — maximum size of an individual token.
- `MaxAuthentications` — maximum possible authenticated authorities.
- `MaxTransactionLimit` — maximum number of active transactions.
- `DefSessionTimeout` — default length of session timeout in miliseconds.

There are also defined properties which are not required in Opal, as they describe features that are not mandated:

- `MaxReadSessions` — maximum number of reader sessions.

- `MaxAggTokenSize` — maximum size of a combined token.
- `MaxSessionTimeout` — maximum possible session timeout.
- `MinSessionTimeout` — minimum possible session timeout.
- `DefTransTimeout` — default transaction timeout.
- `MaxTransTimeout` — maximum possible transaction timeout.
- `MinTransTimeout` — minimum possible transaction timeout.
- `MaxComIDTime` — timeout for ComID.
- `ContinuedTokens` — support of continued tokens (token with size unknown at their beginning, consist of several tokens, each carriying part of the final value).
- `SequenceNumbers` — support of packet numbering (in Packet).
- `AckNak` — support of ACK/NAK (in Packet) in communication.
- `Asynchronous` — support of asynchronous communication.

maybe introduce packetization earlier?

Other than the defined properties, the level 1 Discovery process may also report other, vendor specific, properties.

3.3.3 Level 2 Discovery

Level 2 Discovery is the act of reading a table and is provided by the Get method of an SP. This includes reading any table, such as the access control table or table of locking ranges. Some of the SPs' important tables are described in later chapter 1.

3.4 Life cycle

In order to keep information about the state and the working capacity of an SP, the concept life cycle is introduced. The life cycle describes the condition of an SP using one of several states. In the Core standard, the following states are introduced:

- "nonexistent" — the SP does not exist. The SP might not have been created or already deleted.
- "issued" — the SP is in functional state.
- "issued-disabled" — the SP can only be authenticated to and enabled. All other functionality is disabled. An SP can be enabled and disabled using the SPInfo table of the corresponding SP.
- "issued-frozen" — no functionality of the SP is enabled. An SP can be frozen and unfrozen using the SP table of the Admin SP.
- "issued-disabled-frozen" — the SP is both disabled and frozen as described in the two previous items.
- "issued-failed" — SP is in a fatal failure from which it cannot recover.

3.4.1 SP Issuance

SP issuance is the process of creating a new SP from the Base Template and a set of other templates. This is achieved by calling the Admin SP's method IssueSP. After an SP is issued, it can be personalized. Personalization of an SP is the process of creating new tables (using the method CreateTable), filling in existing tables (using methods CreateRow and Set), and setting up access control (using the method AddACE).

3.4.2 Life cycles in Opal

Opal extends the states defined in the Core specification with a new set of life cycle states called "manufactured" — "manufactured-inactive", "manufactured", "manufactured-failed", "manufactured-disabled", "manufactured-disabled-frozen", "manufactured-disabled". Each of these new states mirrors the corresponding "issued" state from the Core specification. Compared to the "issued" states, the "manufactured" states are created during manufacturing by the manufacturer instead of during the subsequent use by the TPer owner. This is reflected by the fact that these manufactured SPs are not issued and

deleted in Opal. Instead, they are activated and reverted. Activation automatizes the personalization using hardcoded, preconfigured values. Reverting SP maintains its existence, returning it to the factory state instead. Note that this means that the "manufactured-inactive" corresponds to the "nonexistent" state.

3.4.3 Taking ownership

In order to take ownership of a Opal drive, there are several approaches defined in the Opal standard [8]. The most frequent approach is to store the default PIN of the SID authority in a table accessible without any credentials. This allows anyone to read the default PIN, authenticate themselves as the SID authority, and change the PIN. However, the process of taking ownership is not required in any way, and the drive will function as expected even if the PIN remains unchanged.

this doesn't really belong here, but where would i belong?

this is introduced later in access control chapter....

3.5 Communication

In order to send commands to the TPer or the SPs and receive responses, a specific protocol must be used. The communication protocol is split into several layers.

Rephrase this..

Starting from the lowest layer described in this thesis is the interface layer. The interface layer corresponds to the communication with the control unit of the disk. Depending on the disk interface, there can be different security commands used to communicate. The Core standard abstracts these commands into two commands: IF-RECV and IF-SEND for receiving data from disk and sending data to disk, respectively. These commands facilitate all the communication necessary to communicate with the higher layers and are described more closely in section 3.11.2.

The following layer is the TPer layer, a layer used primarily for the allocation of ComID. Since on this layer, the ComIDs are not used yet, and so the state of the communication cannot be kept, the communication is only one way, each "session" consisting only of one command.

The Communication layer supports ComIDs and therefore allows two-way communication and is used primarily for further ComID management, such as getting the state of ComID or resetting the state of ComID.

The Management layer facilitates establishing of a session between an SP and the host. Since the session number is not issued yet, this layer uses Control Sessions with a static session number.

The final layer is the Session layer. At this layer, the session is already established, and communication can be performed using methods sent in packets. Most of the communication occurs in this layer with the use of method invocations.

3.5.1 ComID

One of the information required in order to send command is the ComID, a number identifying the caller (e.g. application of the host). It ensures that responses to method calls will be received by the correct application since there can be at one time multiple callers. There are three types of ComIDs: statically allocated ComID, dynamically allocated ComID and special ComIDs. Statically allocated ComID can be acquired through the level 0 Discovery process. The Discovery process simply informs about the base ComID number and the number of static ComIDs. Dynamically allocated ComID can be acquired using the `GET_COMID` command. There also exist special ComIDs that are used to invoke the Level 0 Discovery or for ComID management.

Every ComID is in one of a few states. These states are Inactive, Issued and Associated. If a ComID is in an Issued or Associated state, it is considered Active. After `GET_COMID`, a ComID becomes Issued, and if a session is opened under it, it becomes Associated. An associated state becomes Inactive again once all sessions opened under it have been closed. The Opal standard does not require dynamically allocated ComIDs to be supported. In Opal, statically allocated ComIDs are always Active.

3.5.2 Packetization

The communication in the last two layers is based on packets. There are three packet types that are nested, each packet type providing a

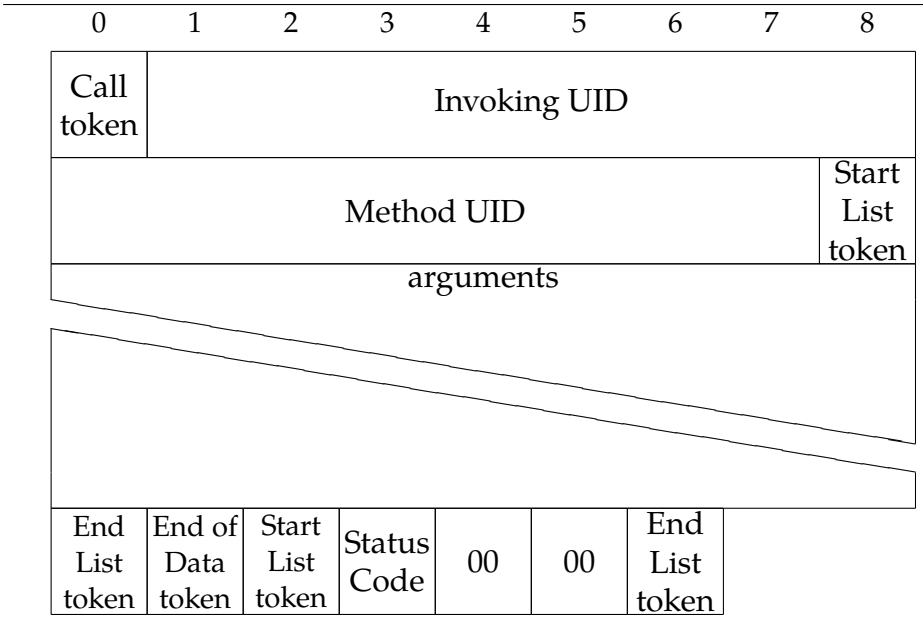

Figure 3.1: Format of method invocation

different service. ComPacket is the outermost one. Each ComPacket contains data for communication under only a single ComID. Each ComPacket can contain several Packets. The usage of Packet is intended for session reliability, specifying sequential numbers and the acknowledgements for them. Every Packet then consists of one or more SubPackets. Finally, SubPacket may be either of Data type or Credit Control type. Data SubPacket contains one or more method calls or results of method calls. Credit Control Subpackets may be used for buffer management to inform the other communication party about the remaining buffer amount.

3.5.3 Methods

The commands of the lower layers to TPer and SPs and their respective responses are transported by the Data SubPackets in the form of methods. These are expressed using a set of tokens in a specific order, described by figure 3.1. Other than the constant tokens such as call tokens or end list tokens, each method invocation contains Invoking UID, Method UID, arguments, and status code list. Invoking UID is used to identify which structure the method is invoked on. Other than

maybe describe tokens? but that seems like a little bit too much,,, right?, maybe just simple, sequence, control

tables and objects, there are also special Invoking UIDs which can be used. These special Invoking UIDs are ThisSP, signifying the SP with which the current session is open, and SMUID, signifying the session manager. Method UID specifies the method to be called. Since SMUID methods are used for managing sessions, when the host is not connected to any of the SPs, SMUID methods cannot be found in the MethodID table. Each method specifies its type of Invoking UID, whether it is a table, an object, or any of the special UIDs, and its parameters. There are two kinds of parameters: the required parameter, identified by its position, and the optional parameter, used in a structure using an integer to identify the parameter, followed by the data of the parameter.

define object as a row of a table

move this to later part since it stands out like a sore thumb

There are two types of arguments: required arguments that all have to occur in the specified order, and optional arguments, which can be left out. The optional arguments are identified by a preceding unsigned integer.

The last 5 bytes of the Method invocation are called Status List. It is a list consisting of a Status Code and two reserved bytes defined to be 0 bytes. A non-zero Status Code corresponds to an error response such as insufficient space, frozen SP, or usage of a method without sufficient authority.

The method structure is used not only for the invocation of methods but also for the response, which uses parameters to return data.

actually, usually not true. usually the call token and the 2 uids are not there, it's there only for syncsession etc, where the response is different method i guess

3.5.4 Sessions

Whereas ComID is used to differentiate the senders of the method and the recipients of the method responses, sessions are used for the parallelization of communication of one such actor. Each session can have different authorized authorities and methods in process, but each session is still bound to one ComID. Since each ComID corresponds to one user, both the user and the TPer can send multiple methods calls or responses in one ComPacket. Sessions use a system of readers-writer locks to enable several concurrently running read sessions without causing concurrency issues. Each session is identified by it's session number. The session number consists of the TPer session number selected by the TPer and the host session number selected by the host.

maybe move closed to ComID?

not user or actor but host... e.g. if it would be network storage, there can be some thingie that would split the communication

There are two types of sessions: control session, used for session management, and regular session. Since the control session is used to establish a regular session with an SP, it is not connected to any SP. The control session is used for managing the regular sessions and, as such, provides methods such as:

- The Properties method enables to find a compromise between the communication capabilities of the host and the TPer or simply find the communication capabilities of the TPer. Using this method, the properties defined earlier in section 1, such as maximum packet size, are established.
- The StartSession and SyncSession methods provide a way to start an unencrypted session.
- The StartTrustedSession and SyncTrustedSession methods provide a way to complete challenge-response authentication or to setup secure messaging/key exchange.
- The CloseSession method used to close the session.

Each ComID has one associated control session with a lifespan the same as the ComID, since a control session is required to create regular session. On the other hand, there can be multiple regular sessions.

Depending on the authorities used, it is also possible to start a secure messaging session, which provides message authenticity and/or confidentiality. Authority's operation type decides the what kind of trusted messaging is to be used. These are further described in the subsection 3.6.1. However, Opal does not mandate the support of secure messaging.

3.5.5 Transactions

In order to facilitate a safe execution of sequences of methods, the Core standard specifies transactions. Similarly to transactions in database systems, this feature enables one to revert the effects of a sequence of methods. This is done automatically in case the transaction is not finished or if the transaction is manually aborted. However, not all the effects of methods are rolled back, such as logs. Nested transactions are supported, in which case the transaction is committed when the outermost transaction is finished.

3.5.6 Memory structures

In order to maintain the state of the device, tables are exclusively used. Tables are defined by their columns, and each row is called an object. Each template defines its tables, such as the key tables containing keys for different LRs, or the meta table called the Table table, which contains information about all the other tables in the SP. Although the Core standard specifies a way to create new tables, or delete existing ones, this feature is not required by the Opal standard.

3.6 Access Control

In order to provide access to methods only to authorized actors, the standard also defines access control. The access control provides a way to allow access to methods of the SP only after the knowledge of a secret has been proved. Information required to verify knowledge of the secret is called a credential and is stored in one of the corresponding SP's credentials tables. Each SP may contain several credential tables, one for each type of credential, such as a password, an RSA key pair, or an AES key.

In order to support the authentication of different users or require multiple users at once, access control rules are defined using Access Control Lists (ACL) containing Access Control Elements (ACE). ACEs are Boolean expressions with inputs being authentication of authorities.

Each combination of method and object has a defined ACL, which needs to be satisfied in order for the method to be invoked on the object. An ACL is satisfied once one of the ACEs contained within it is satisfied, where authenticated authority variables evaluate as true.

Other than explicit authentication, implicit authentication is also possible. Implicit authentication may be used with, e.g. the Exchange operation type where the session key by the SP is encrypted using a pre-shared key and returned to the host.

Authority may be authenticated either during session startup using parameters of the SessionStart method, or, if there is more than one authority, the additional authorities may be authenticated using Authenticate method. The Authenticate method authenticates only using explicit methods. In case Authenticate method is used to au-

thenticate using a password, one invocation of the method with a password as the parameter is enough. In the case of challenge and response operation type (Sign, SymK, HMAC), the first invocation only specifies the authority to be authenticated and receives challenge in response, and the second invocation provides the response.

Each authority is an object in the authority table of the corresponding SP and has a type which is either individual or class. Class authorities correspond to a set of authorities so that they can be easily assigned and changed in bulk.

3.6.1 Authorities

Authorities are constructs used to represent an actor. Authorities are saved in the Authority table of every SP, provided by the Base template, and so are separate for each of the SP. Each authority is associated with one credential. This credential is an object from a table, depending on the type of the credential.

The Base template defines several optional authorities such as the Admin authorities, representing the owner of the SP, the Makers authorities, representing the manufacturer of the TPer, the Security Identifier authority (SID), representing the owner of the TPer, and TPer authorities, representing the TPer itself. The Base template also defines a special authority Anybody without any credentials. This authority can be used in reading public info of the SPs or to take ownership of the TPer.

Each authority has assigned one operation type. This operation type determines how can this authority be used to authenticate. Some of these types are Sign, used to authenticate using asymmetric cryptography, HMAC, used to authenticate using challenge and response with HMAC, and Exchange, used to share session key encrypted using the authority's credential. However, the only operation type mandated by Opal to be implemented is the Password type, allowing to get explicitly authenticated using a password. This means that any other way of authentication such as implicit authentication, and subsequently also secure messaging, may not be available.

3.7 Templates

Since SPs may share some of their functionality such as authentication, modification of tables, or retrieval of data from tables, there exists templates to define this shared functionality. Each of the SP then implements a subset of functionality of one or more templates. Since templates

The Core [9] standard defines the following templates:

- *Base Template* defines the shared subset required by every SP and is therefore mandatory. The most important functionality that is defined by this template is access control and metadata.
- *Admin Template* is a template specific to the unique Admin SP. It provides access to methods managing the TPer.
- *Crypto Template* provides methods providing cryptographic methods, such as encryption, hashing, or signing.
- *Locking Template* is a template specific to the unique Locking SP. Provides access to methods managing the lock state of the device, and locking ranges.
- *Clock Template* provides methods for indicating current time, measuring lag, and access to monotonic counter.
- *Log Template* provides methods for logging activity. The logging can be either carried out manually by an user (or an user application) or automatically by the TPer (as a result of invocations of methods of SPs containing this template, including invocations during read-only sessions). Only one system log table may exist on each SP, but multiple user log tables may be present on one SP. The logs are saved on non-volatile storage, depending on the security setting of the table, they may be either buffered or be saved after each entry.

emph at start of sentence or em dash? em dash is not typographically correct though....

how much are the logs preserved... will they get deleted with an SP?

Even though the Core standard defines several templates, Opal does not mandate them. Due to the fact that issuance of SPs is optional in Opal and that the only mandatory SPs are the Admin SP and Locking SP, Opal devices need to implement only the Base Template, Admin Template and Locking Template.

3.8 Admin SP

Admin SP is one of the two SPs mandated by the Opal standard [8]. It is a unique SP, which holds information about the TPer and all the present SPs. It allows creation, deletion and general modification of the SPs. It is created from Base and Admin Template. This SP is initialized in the "manufactured" state, so that it can function as a starting point of TPer management.

we describe the methods and tables as defined for opal.

3.8.1 Methods

Even though the templates from which the Admin SP consists define many more methods, the Opal standard mandates only the following ones:

- Base Template methods:
  - "Next" — returns UIDs of objects in the table. Allows to specify the returned amount and the first UID.
  - "GetACL" — returns the ACL of the method and object or method and table combination.
  - "Get" — returns the value of the object or the contents of the byte table.
  - "Set" — sets the value of the object or the contents of the byte table.
  - "Authenticate" — performs an additional explicit authentication.
- Admin Template methods:
  - "Revert" — reverts the SP referred to by the invoking ID to its initial state.
  - "Activate" — activates the SP referred to by the invoking ID.
- Crypto Template methods:
  - "Random" — returns specified amount of random bytes.

3.8.2 Tables

- Base Template tables:
  - SPInfo — metadata about the SP
  - SPTemplates — templates used by the SP
  - Table — metadata about tables used by the SP
  - MethodID — metadata about methods used by the SP
  - AccessControl — ACLs for method and table combinations
  - ACE — definitions of ACEs
  - Authority — list of authorities
  - C_PIN — password credentials of authorities
- Admin Template tables:
  - TPerInfo — metadata about the TPer, versions, implemented SSCs, free space for issuance, ProgrammaticResetEnable
  - Template — metadata about templates, numbre of instances
  - SP — metadata about SPs, current life cycle, frozen status
- Opal additional tables:
  - DataRemovalMechansim — describes mechanism for data removal, such as overwrite erase, block erase, or cryptographic erase

3.9 Locking SP

Locking SP is the second SP mandated by the Opal standard [8]. It is a unique SP, which takes care of disk encryption, locking, and unlocking, and tasks related to it.

3.9.1 Methods

Locking SP shared the Base Template and Crypto Template methods with Admin SP, and adds some own:

- Locking Template methods:
  - "GenKey" — regenerates a key or a credential. But, in the case of Opal, this method is mandated to be able to be used at least on the `K_AES_*` DEK keys.
  - "RevertSP" — reverts the Locking SP. Compared to "Revert" this method can revert only the SP to which the session is connected, allowing reverting without having access to Admin SP's Admin authority.

3.9.2 Tables

- Extra Base Template table:
  - SecretProtect — specifies protection mechanism for secrets, such as in case of Opal DEKs. Possible mechanisms are binding to C_PIN values, or vendor specific [18].
- Locking Template tables:
  - LockingInfo – metadata for locking mechanisms, such as maximum number of locking ranges, or supported types of encryption.
  - Locking – contains state of locking ranges, more closely described in subsection 3.9.3.
  - MBRControl – metadata for MBR shadowing (Enable, Done, DoneOnReset).
  - MBR – byte table containing the shadow MBR.
  - K_AES_128 and/or K_AES_256 – DEKs.
- Non template tables:
  - DataStore – byte table for storage of arbitrary data

3.9.3 Locking range

The locking range feature gives the user a way to specify an LBA range on the disk that can be locked independently on the rest of the disk.

Each locking range also has its own ACE to control who can lock and unlock the range. Because each LR is also encrypted using its own DEK, it is possible to regenerate new DEK for a single LR, discarding data of only one LR.

There also always exists a special range called the global locking range that covers any area on the disk that is not already covered by a regular locking range.

In order to control the access to the data on a disk covered by a specific locking range, the Locking table contains columns ReadLockEnabled, WriteLockEnabled, ReadLocked, and WriteLocked. The data inside the LR cannot be read when ReadLockEnabled and ReadLocked are both set to true. Otherwise, the LR range is locked, and the data cannot be read. Similarly, for writing and WriteLockEnabled and WriteLocked. The separation of the locking right into the LockEnabled and Locked column gives the ability to have separate ACE for the LockEnabled column, meaning that, for example, an admin can decide that the user can only lock reading for a specific LR but not writing.

Each locking range also has a defined column LockOnReset containing a list of types of resets on which the LR gets locked. For Opal devices reset Power Cycle must be always selected, and Programmatic reset can be selected additionally. Last reset type Opal defines for the drives is the optional Hardware Reset.

maybe what they are?

3.9.4 Single User Mode

In many cases, it might be desirable to prevent admins from accessing user data, even though the admins generally have more competence. The Single User Mode Feature Set [17] defines a way to achieve this. After activating Single User Mode, only a single User authority is capable of changing their authority object (this includes changing their PIN), changing the proprieties of LRs assigned to them (which includes the lock state of the LR) and generating new keys for those LRs. The only actions available to the Admin authorities related to that LR are destructive actions.

3.9.5 MBR shadowing

This feature enables the disk to provide a fake master boot record (MBR). Instead of the MBR saved on the disk, the disk instead provides the shadow MBR saved in a table on bootup. The shadow MBR may contain software to enable the host to authenticate itself to the disk and unlock it. After the host is authenticated, the shadow MBR may be deactivated, and the regular disk data will be available again.

This feature is controlled using tables `MBR`, a byte table containing the data to be presented while the shadowing is active, and `ControlMBR`, an object table used to control the shadowing. Table `ControlMBR` one row with columns `Enable`, `Done`, and `MBRDoneOnReset`. When `Enable` is true and `Done` is false, the shadowing is active, and the data from `MBR` table can be read from the disk. The column `MBRDoneOnReset` specifies a list of types of resets during which the `Done` is set to false.

specify reset types?

3.10 Opal SSC

remove this, but the intro seems nice to have? maybe to introduction?

Even though the Core specification introduces many powerful and interesting features, this might increase the cost of design and manufacturing of a fully compliant device. To solve this problem, the Opal SSC determines only a small set of mandatory features, leaving most of the features optional. Together with the set of mandatory features it also states range limits of certain properties.

For access control, Opal is mandating only password authentication, this means that any other authentication such as implicit authentication of the host or any authentication of the TPer (and therefore also secure messaging) may not be available. For communication Opal requires only support of synchronous communication. For table management, Opal does not require support of creation or deletion of tables. Issuance of SPs is also not required, and Opal instead uses SPs preconfigured by the manufacturer. Out of the previously mentioned SPs, Opal requires only that the Admin SP and the Locking SP are supported. Since SP issuance is not one of the mandatory features, the Locking SP may also be preconfigured by the manufacturer and be initially in the “manufactured-inactive” state.

Due to the range limits specified in the Opal SSC, the following features also might not be available.

- Opal specifies that the device must be able to handle: at least one method per SubPacket, at least one SubPacket per Packet, and at least one Packet per ComPacket. This means that any ...

- At least one active transaction per session, at least one active session per all the ComIDs and at least one ComID.

This means that an Opal-compliant device implementing only the required minimum will also not be able to provide any

But Opal does not only reduce the feature set from the Core standard, but also expands it with some extra feature sets. Even though the Core specification does not require the following feature sets, the Opal SSC requires them additionally.

- Physical Presence SID (PSID) is a special authority that is authorized to call only the Revert method. The PIN for this authority may not be found out using the interface of the disk, every one of our disk supplied this information on a label on the disk. This authority provides a way to reset the TPer into factory state even in case the SID PIN is lost [16].

- DataStore tables are byte tables accessible for any use. Using the Activate method, the number of the DataStore tables with their sizes can be specified [13].

- Block SID Authentication feature disables SID authority until device restart. One of possible use cases is by BIOS to protect a drive with default MSID [14].

From our experience with Opal disks, most of them did not implement more than the required minimum. Some of the tested disks, even though they were described by the vendor and/or manufacturer as Opal-compliant, did not implement every required feature set. Out of the 6 tested disks, only 2 implemented the required Block SID Authentication feature set. However, even those 2 implementing the feature set did not support the actual feature.

TODO: fact recheck later on

3.11 Host-side implementation

The Linux kernel offers two ways to control an Opal device from the host. Both ways use the Linux ioctl system call. In this section, we will introduce these two approaches

3.11.1 Opal ioctl

Since version 4.11, the Linux kernel offers a set of ioctl requests to facilitate control over an Opal disk [19]. The individual implemented ioctl requests for Opal functionality and their function as of Linux kernel 5.19 are:

- `SAVE` — keeps a key for a locking range in the host's RAM so that it can be used after waking the disk up from being suspended. To wake the disk up, exported symbol `opal_unlock_from_suspend` can be used to unlock the disk with the saved data.
- `LOCK_UNLOCK` — locks or unlocks reading or writing for the selected locking range.
- `TAKE_OWNERSHIP` — changes the admin authority password from the default one to the selected one.
- `ACTIVATE_LSP` — changes state from "Manufactured-Inactive" to "Manufactured" state. Also facilitates the setup of Single User Mode.
- `SET_PW` — changes the password of the selected authority using the admin password.
- `ACTIVATE_USR` — enables the specified user in the Opal tables.
- `REVERT_TPR` — reverts the TPer to the manufactured state using the admin password.
- `LR_SETUP` — sets locking range position/locking enable
- `ADD_USR_TO_LR` — sets the ACE for locking and unlocking to the designated user. Contrary to the name, this request does not actually add a user and instead replaces any existing one by the new one.

write about the initialisation of the TPer to the generic chapter, C_PIN_MSID etc.

write about TPer states in the generic chapter.

- `ENABLE_DISABLE_MBR` — changes the `Enable` column of the `MBRControl` table.
- `ERASE_LR` — calls the erase method defined in Single User Mode Feature Set [17]. This method not only destroys the data in the locking range, but also resets the assigned user authority of the locking range and the users password. This is required since a Single User Mode protects these values from change by authorities other than the assigned user.
- `SECURE_ERASE_LR` — regenerate the data encryption key of a range to destroy the previous one.
- `PSID_REVERT_TPR` — resets the TPer to the manufactured state using PSID.
- `MBR_DONE` — changes `Done` column of the `MBRControl`.
- `WRITE_SHADOW_MBR` — writes data into the MBR table.
- `GENERIC_TABLE_RW` — reads a byte table or writes into a byte table. Neither object tables nor iteration is supported.

Although these ioctl requests offer simple access to control the Opal capabilities of a disk, not every feature mandated by Opal is implemented. Some of the limitations are the following:

- Access to only a single admin authority and up to 9 user authorities.
- No way to change the password of SID authority. This means that the authority representing the owner of the device, which often has control over the entire disk, is stuck with the credentials chosen during the taking of ownership.
- No write-only lock.
- Each locking ranges can be configured with only a single user capable of locking and unlocking it.
- No way to read or write to object table rows or iterating tables. This prevents the possibility of replacing the missing features using a direct setting of values in a table.

Currently, there is no documentation to be found for the ioctl requests for Opal functionality. The information in the previous list was acquired from our code analysis. Short program showcasing the usage of this interface can be seen in the appendix 1.

3.11.2 Drive interface ioctl

Alternative to the Opal ioctl requests are the disk controller ioctl requests. Depending on the disk protocol, a different way of passing the Opal commands is required. The TCG Storage Interface Interactions Specification [20] describes the mapping of the abstract IF-RECV and IF-SEND commands to the actual commands of disk interfaces for different drive interfaces.

For NVMe drives, there exists the `NVME_IOCTL_ADMIN_CMD` ioctl request to send commands to the disk, which uses the `nvme_admin_cmd` structure [21]. The IF-RECV and IF-SEND are mapped to the Security Receive and Security Send NVMe's command respectively [20]. The structure specifies the Security Protocol, the ComID, and the data to transmit to or from the TPer.

describe security protocol somewhere??

For ATA drives, the situation is more complex. There does not exist an ATA command we could use to send the Opal commands. However, using the SCSI `SG_IO` ioctl request with it's `sg_io_hdr_t` structure, we are able to use the ATA PASS-THROUGH command [22]. Using the ATA PASS-THROUGH command, we are able to send arbitrary ATA commands to the device, including the TRUSTED RECEIVE and TRUSTED SEND commands [3] mapped to IF-RECV and IF-SEND respectively.

Using these structures, the Opal commands described in section 3.5 can be sent to the TPer. Compared to the Opal ioctl requests, this has the advantage of not being limited only to a subset of features that the Opal ioctl requests implement and instead being able to use every Opal feature the device offers.

Although this approach gives the host application access to every feature of the Opal disk, it also requires it not only to implement the command hand-over for each type of disk separately but also to create the methods and parse the method results, both described in the chapter 3.5, on their own.

disk interface popsat[NVME]

4 Existing tools

In order to give users the ability to manage their SED disks without the need to create their own programs to access existing interfaces, there exist several tools.

In this chapter, we will look at several existing tools that allow one to manage the encryption of disks.

4.1 Cryptsetup

Cryptsetup[1] is a tool for disk encryption setup. Although this tool supports several formats and volumes, such as ..., all of them are software, and there is no hardware encryption support as of now.

probably remove this since it does not support any hardware encryption,,, yet

4.2 hdparm

The *hdparm*[2] tool provides a command line interface to control parameters of ATA disks. Among others, this tool gives access to control ATA Security Feature Set. The examples of SED-relevant commands for the functionality described in 1 are:

- Enable the ATA Security Feature Set:
  `hdparm --security-set-pass pwd /dev/sda`
- Disable ATA Security Feature Set:
  `hdparm --security-disable pwd /dev/sda`
- Unlock the disk:
  `hdparm --security-unlock pwd /dev/sda`
- Erase the disk:
  `hdparm --security-erase pwd /dev/sda`

Implicitly the user password is referred to in the commands, but `--user-master` can be specified to select master or user password explicitly. While enabling the ATA Security Feature Set, it is also possible

1. `https://gitlab.com/cryptsetup/cryptsetup`
2. `https://sourceforge.net/projects/hdparm`

to use the `--security-mode` option to choose between high and maximum security mode. A command to lock the drive is missing because a drive with enabled ATA Security Feature Set is locked when powered off.

4.3 sedutil

sedutil[3] is a tool for management of SED disks, maintained by the Drive Trust Alliance. It currently supports the Enterprise and Opal SSCs (and additionally the remaining Pyrite, Opalite and Ruby SSCs in a fork of the project[4]). Even though it is currently the largest open-source project to control SED disks, the code repository does not seem to be currently active.

This tool does not use the Linux Opal ioctl interface and instead uses the approach described in the subsection 3.11.2. This allows the tool not only to be multi-platform but also to use the Opal features in their entirety instead of only the subset introduced in the Opal ioctl interface.

The tool offers the following commands, grouped by their functionality:

- Discovery:
  - `isValidSED` — lists SSCs supported by the selected disk.
  - `scan` — performs `isValidSED` command on every disk in the system.
  - `query` — performs Level 0 and Level 1 Discovery on the selected disk.
  - `printDefaultPassword` — prints the MSID password.
- TPer management:
  - `initialSetup` — takes ownership of the disk, and initializes Locking SP, global LR and shadow MBR.
  - `setSIDPassword` — changes the password of SID authority.

3. `https://github.com/Drive-Trust-Alliance/sedutil`
4. `https://github.com/ChubbyAnt/sedutil`

  - `setAdmin1Pwd` — changes the password of Locking SP's first admin authority.
  - `setPassword` — changes the password of any supported Locking SP's authority.

- Locking range management:
  - `listLockingRanges` — prints information about all LRs,
  - `listLockingRange` — prints information about a single LR,
  - `rekeyLockingRange` — regenerates the LR's key (destroying the data in the LR in the process)
  - `setupLockingRange` — sets the start and length of locking range,
  - `setLockingRange` — sets ReadLocked and WriteLock of the LR depending on the chosen configuration (read-write, read-only, or locked).
  - `enableLockingRange` — sets ReadLockEnabled and WriteLockEnabled of the LR.
  - `disableLockingRange` — unsets ReadLockEnabled and WriteLockEnabled of the LR.

- Shadow MBR management:
  - `setMBREnable` — sets Enable column of the MBRControl table.
  - `setMBRDone` — sets Done column of the MBRControl table.
  - `loadPBAimage` — writes the contents of a file into the MBR table to be used as the shadow MBR.

- Reverting the device:
  - `revertTPer` — reverts the TPer using the MSID password.
  - `yesIreallywanttoERASEALLmydatausingthePSID` — reverts the TPer using the PSID password.
  - `revertNoErase` — reverts only the Locking SP, keeping the global range data.

- Enterprise-specific functionality:
  - `setBandsEnabled` —
  - `setBandEnabled` —
  - `eraseLockingRange` —

4.4 go-tcg-storage

The *go-tcg-storage*[5] is a Go library that does everything, and is actually supported. It has a few tools that let you do some things. It also supports more SSCs than others. Just found out about it, will need to look into it deeper and figure out why is it worse, so we won't look as bad.

CLI: Was able to generate random numbers, but only for some of the disks.

4.5 TCGstorageAPI

Seagate's *TCGstorageAPI*[6] is yet another

Provides a Python library, written in c++, and a CLI...

Has a bunch of dependencies, most importantly boost, which is pretty big...

CLI: doesn't work with sdc "SED configuration is Unknown/Unsupported (Type Opal) - Exiting Script" no random numbres

offers test suite!!!

4.6 fscrypt

probably rework into text about the tool for management instead?

4.6.1 Usage

As of now, block layer inline encryption is supported only by two file systems in Linux: ext4 and F2FS. In this part, we will focus only on the ext4 filesystem.

requires the filesystem to actually use the library In the following paragraphs, that should be redone, I am assuming ext4.

5. `https://github.com/open-source-firmware/go-tcg-storage`
6. `https://github.com/Seagate/TCGstorageAPI`

First of all, the inline encryption needs to be enabled in the kernel configuration. This means that the Linux kernel configuration option `CONFIG_FS_ENCRYPTION_INLINE_CRYPT` needs to be enabled. In order to start using the file system with encryption, it needs to be mounted with the option to specify the usage of inline encryption.

```
mount -t ext4 /dev/foo /mnt/foo -o inlinecrypt
```

It is not enough to specify the inline encryption flag. The encryption itself also must be enabled in the file system. On ext4 file systems, the encryption can be enabled after mounting like so:

```
tune2fs -O encrypt /dev/foo
```

After this, fscrypt can be used as normal and it will use inline encryption for this filesystem.

In order to encrypt a folder using the kernel space fscrypt, the following must be done: an encryption key must be added and the encryption policy must be created. In C, this can be done the following way:

```
int fd = open(pathname, O_RDONLY | O_CLOEXEC);

// add a key
struct fscrypt_add_key_arg *key_request = calloc
   (1, sizeof(struct fscrypt_add_key_arg) +
   key_len);

key_request->key_spec.type =
   FSCRYPT_KEY_SPEC_TYPE_IDENTIFIER;
key_request->key_id = 0;
key_request->raw_size = key_len;
memcpy(key_request->raw, key, key_len);
ioctl(fd, FS_IOC_ADD_ENCRYPTION_KEY, key_request
   );

// set a policy
struct fscrypt_policy_v2 policy_request = { 0 };

policy_request.version = FSCRYPT_POLICY_V2;
```

```
policy_request.contents_encryption_mode =
   FSCRYPT_MODE_AES_256_XTS;
policy_request.filenames_encryption_mode =
   FSCRYPT_MODE_AES_256_CTS;
policy_request.flags =
   FSCRYPT_POLICY_FLAGS_PAD_8 |
   FSCRYPT_POLICY_FLAGS_PAD_16 |
   FSCRYPT_POLICY_FLAGS_PAD_32;
memcpy(policy_request.master_key_identifier,
   key_request->key_spec.u.identifier,
   FSCRYPT_KEY_IDENTIFIER_SIZE);
ioctl(fd, FS_IOC_SET_ENCRYPTION_POLICY, &
   policy_request);
```

This code sets up an encryption policy for the file specified by the pathname.

4.6.2 Implementation

maybe remove completely?, doesn't seem to fit much...

During mounting the "inlinecrypt"/`SB_INLINECRYPT` flag is written into the `super_block` structure.

It all starts in `__ext4_new_inode`. This is the internal function used when creating new inodes, called by functions such as `ext4_create` when creating a new file.

The function (if it is not inode used for large extended attributes?) calls `fscrypt_prepare_new_inode`.

`fscrypt_prepare_new_inode -> fscrypt_setup_encryption_info -> setup_file_encryption_key -> fscrypt_select_encryption_impl`

In `fscrypt_select_encryption_impl` there is actually the only place where the `SB_INLINECRYPT` flag is used. ... Calls `blk_crypto_config_supported` to check the device's crypto profile. Afterwards, `fscrypt_select_encryption_impl` function sets the `(fscrypt_info *)ci->ci_inlinecrypt`.

`setup_per_mode_enc_key` then sets the `(fscrypt_info *)ci->ci_enc_key`.

fscrypt - How does it work? - Usage The bio function is stored in `(struct bio *)bio->(struct bio_crypt_ctx *)bi_crypt_context`

Function `fscrypt_set_bio_crypt_ctx` changes the file's bio to use inline encryption... simply calls the blk layer `bio_crypt_set_ctx`.

Calling `submit_bio` like normally ... `__submit_bio` calls `__blk_crypto_bio_prep`...

"If the bio crypt context provided for the bio is supported by the underlying device's inline encryption hardware, do nothing."

`__blk_crypto_rq_bio_prep` however sets the context of the request to the one of the bio... After the bio prep `blk_mq_submit_bio` gets called (which calls `blk_mq_bio_to_request`, and after that also `blk_crypto_init_request->blk_crypto_get_keyslot` which updates the devices keyslot to contain the new key..., but does nothing if the device does not have keyslots)..... the info about the key to use then has to be acquired by the driver from `request->`

Most important are probably structures `blk_crypto_ll_ops` and `blk_crypto_profile`... just two operations, program key and evict key.

how to get crypto profile from outside..

Where does the hardware come to play?

`ufshcd_exec_raw_upiu_cmd()->ufshcd_issue_devman_upiu_cmd()->ufshcd_prepare` sets the header with the correct keyslot.

4.7 Proprietary solutions

Some disks might provide their own software for their own proprietary hardware encryption. These are usually found separately on website or something, but some are provided as by the disk on an unecrypted partition.

5 Security of hardware encryption

Self-encrypting disks usually do not ... For example, the Opal standard aims to only "Protect the confidentiality of stored user data against unauthorized access once it leaves the owner's control (following a power cycle and subsequent deauthentication)" [8]. Even if the device is successfully protected against such a threat model, it still leaves many possible attack vectors. To extend the threat model into a more realistic scenario, we consider an attacker that has physical access to the disk installed in a "locked" computer or something.

It should be noted that some systems

In order to limit the scope of the analysis, we will focus primarily on the hardware... The threat model we consider in this analysis is primarily going to be the same ...

5.1 Attacks on hardware encryption

In this chapter, we will provide an overview of state-of-the-art attacks relevant to hardware disk encryption. For each attack, we will provide our theoretical analysis of the protection offered by Opal, if properly implemented in a disk, against such an attack. For a selection of the attacks, we will also provide our practical insight.

TODO: clean up with citations from the vulnerabilities papers[23, 5, 24, 25, 4, 26].

5.1.1 Evil maid attack

An evil maid attack consists of a situation where the device is left unattended, and the attacker has full physical access. Such attacks usually have two phases. In the first phase, the disk is modified, e.g. by installing custom firmware or inserting a probe to eavesdrop on the communication. Then, the victim uses the disk, providing the password to unlock it. In the second phase, the attacker returns to collect the obtained information and undo their changes. In some cases, the attacker might not require physical access to the device and instead use a network to receive information and have, e.g. firmware reverse itself (or leave it be if the discovery of the attack is not a problem).

Hmmm, maybe something like: Opal does not specify anything about firmware update, and shadow MBR update requires authentication. Still, one could still listen to the cables, I suppose because Opal does not mandate secure messaging and so the communication can be simply sniffed out on the ATA/NVMe/whatever cables?

5.1.2 Attack on RNG

Attack on RNG is such an attack which abuses RNG generating data with low entropy or even entirely predictable.

Since Opal specification does not actually specify any implementation for the RNG, we cannot perform any fruitful analysis of the standard itself. However, we can still perform practical analysis.

The Opal specification offers a few ways how to generate random data. The most useful would be GenKey used for generating DEKs, but since the destination cells are protected from reading, we cannot get to them easily. Another method of generating random data is the method Random. This method allows us to get at least 32 bytes [8] every invocation. Using this method, we have decided to generate random bytes to inspect later. From the disks we have inspected we have found the following:

- SanDisk X300s — first Random invocation of session always return the same bytes. The value of the first bytes seems to change only after the new activation of Locking SP. Following invocations of the Random method in a session return different bytes.

- Kingston KC600 — for the minimum 32 bytes, the TPer closes the session on Random invocation. The same response was for most of the other tested values. However, if exactly $256^n - 1$ bytes are requested, $128 + n$ bytes are returned... lol, I see where the problem is now. The TPer doesn't parse it as a token but as a regular "C" unsigned integer. So, it was reading the header instead, and the `0xff` bytes were required for "NOPs" since they are tokens for empty atoms. With a slight modification, we can get up to 255 bytes from this disk at once.

- Samsung disks — return seemingly random values.

The results of dieharder's Diehard test suite found no statistical something problem. Note that for the Kingston KC600, we did not include the first batch of random bytes.

SanDisk X300s — However, using GenKey on a locking range does change the data into random bytes, so it does not seem this affects key generation.

5.1.3 Attack on interface

An attack on the interface is such an attack where problems in the interface are taken advantage of. This might include attacks such as using vendor commands to read the disk directly, even if the disk is unlocked... In the past, many hardware encrypted disks were vulnerable to such attacks, requiring only the usage of undocumented vendor commands to read either the data of the encryption module (in some cases containing the unencrypted DEK) or even the content of the disk itself.

As can be seen from the previous subsection 5.1.2, the implementation of the interface is not perfect for some of the disks. Something like non-security issues might very likely be a sign of possible security issues, so don't take those problem that we found lightly! Maybe also something that more information can be found in the data chapter or something like that.

TODO: rethink the subsection name.

5.1.4 Cold boot attack

In this attack, the property of volatile memories is used. Even though volatile memories lose the data stored in them after powering off, they do not lose the data instantly. This introduces the cold boot attack, which takes advantage of this fact. One of the ways is to take a RAM stick from the victim's locked computer and install it into the attacker's computer, where it can be read. Another possible approach is the reboot the victim's computer into the attacker's system.

Even though this attack works quite well against software-based disk encryption and software encryption in general, against purely SED, it does not work that well. This is because, compared to software encryption, there is no need to store the encryption key in the

computer's memory. The key is instead stored in the memory of the disk controller, where it can (hopefully) be well protected against removal... (and also, it's not a generic RAM stick). However, this advantage holds only for the case where the key or the disk password is not kept in the computer's memory, and this may not always be the case with SEDs. In the Linux Opal ioctl commands introduced in an earlier chapter, there is introduced functionality to save the password for locking range in the memory. This functionality is introduced in order to allow the computer to wake from suspension automatically since the disk might need to be unlocked in such a case. But since the key is stored in the computer's memory, it is possible to acquire it using the cold boot attack.

5.1.5 Hot plug attack

Hot-plug attacks [27] are attacks similar to cold boot attacks. However, compared to the cold boot attacks, where the RAM is inserted into the attacker system before the data in it is naturally destroyed, hot plug attacks instead move the disk into the attacker's system. This attack abuses the fact that SATA disks have a data cable and a separate power cable. Because of this, it is possible to switch the data cable into a different device without powering off and therefore locking the disk.

In the Core standard [9], there exists a reset type called HotPlug. Although this reset type is not described closer than the name in the standards, it most likely refers to the ATA/NVMe? hot plug TODO[**TODO**]... Using this reset type, it could possible to set an LR to lock when a hot plug is detected. However, as this reset type is not mandatory (nor optional) for Opal devices, and none of our Opal devices implemented it, we were not able to examine this solution further. However, even if the device would implement the HotPlug reset type, using it to prevent the hot plug attack would not be advisable as the hot-plugging process could be disabled in the attacker's system.

6 Opal toolset (name WIP)

implement proof-of-concept low-level tools to access available devices,

In order to be able to study the disks at a closer level, we have implemented a toolset for controlling and inspecting Opal devices. We have focused on implementation that would not require any additional tooling or libraries and would allow as much control over the communication with the Opal device. This approach allows the project to be not only used on a system without introducing a plethora of additional packages but also for our code to be used in other projects without needlessly increasing the number of dependencies.

The code base is split into three levels. Each level implements a different part of the functionality that depends on the previous levels.

The lowest level is focused abstraction of the communication primitives. This includes primarily functions to build a method to be sent to the device and parse the received response, generic functions to start a session or set a value in a row of a table, and the implementation of IF-SEND and IF-RECV commands abstracting ATA and NVMe system calls.

The middle level joins the lower-level methods into complete functions. Each function represents one of the methods and provides a C interface to invoking the method. Internally the functions start a session with the device, build the method according to the function's arguments, send the method to the device, retrieve the response, and parse the response, to find whether to command was successful and to potentially return the data to the caller.

The top level combines the two previous levels to create a usable front end usable through a command line. Currently, there are three front ends: `rng` to produce random numbers generated by the selected Opal disk, the `control` to manipulate the status of the selected OPal device, and the `discovery` to inspect the selected Opal device.

`rng` The `rng` is the simplest front end of the introduced three. Its only goal is to produce random numbers generated by the disk quickly. However, all the disks we have tested implemented only the minimal Opal requirements, which guarantee a limit of only one active session, packets containing only one method, and Random method providing only 32 bytes at once. Such properties do not leave much space for

optimisation. The throughput this utility can achieve with the disks tested by us is between around 500 bytes per second and 35000 bytes per second. Although to obtain a sufficient amount of data, a considerable amount of time is required, this utility may be used to collect random numbers for testing of randomness.

`control` The `control` utility provides access to basic control over the device. As of right now, there are the following commands:

- `--unlock` command sets the ReadLocked and WriteLocked columns of a locking range.
- `--setup_tper` command takes ownership of the TPer.
- `--setup_user` command enables a user and sets its password.
- `--setup_range` command enables the locking range together with its ReadLockEnabled, WriteLockEnabled, sets the range of the locking range, and enables specified users to change ReadLocked and WriteLocked of this range.
- `--psid_revert` command performs the Revert method as the PSID authority [16], reseting the device into the manufactured state.
- `--reset` command sends the TPER_RESET command to the device [9].

(also enables programmatic reset, but that's to change, ... maybe)

These

`discovery` Finally, the `discovery` utility provides information about the Opal disk. This utility provides, other than the Discovery information introduced in section 3.3 also other useful information, such as the identify commands of the drive interfaces (Identify command for NVMe devices [21] and IDENTIFY DEVICE command for ATA devices [3]). Level 0 Discovery and Level 1 Discovery information are acquired easily, as they are defined as they may be acquired using a simple IF-RECV command with specific ComID and Protocol ID or using the Parameters method without any parameters, respectively. The Level 2 Discovery, on the other hand, is more complicated.

Level 2 is actually just tables 'AccessControl' and 'ACE'!!!!!, mention that we actually expand the meaning to cover all tablesf

We would like to include not only the AccessControl and ACE table in this process but also other interesting tables, such as the TPerInfo table, containing information about the firmware version, or the SecretProtect table, containing information about the mechanism used to protect the DEKs. Because of this, instead of the definition of Level 2 Discovery presented in section 3.3, we will use a modified version. This modified version will try to include every table in the TPer. However, we have a few limiting factors that may limit the number of tables we will be able to read. First of all, we do not want to change the state of the device. This includes actions such as reverting the device to the manufactured state or taking ownership of it. Even if the state is potentially reversible without data loss, the risk of the disk not working correctly surpasses the potential gain from more information. This might Another requirement is that no authorisation should be required. Instead, we will use the password-less Anybody authority. This again restricts the set of tables that are accessible. However, according to the definitions of access rights in the Opal standard [8], this mainly affects secrets, such as passwords of Authorities or DEKs, and the current configuration of locking ranges, such as their start and length. Such information does not speak about the properties and capabilities of the device itself.

The process we use in order to print the contents of every table is as follows: Firstly we iterate using the Next method over the SP table of the Admin SP to get a list of existing SPs. As long as the Admin SP is not frozen or disabled, this should always succeed since the Admin SP is in the manufactured state by default, and its SP table is always accessible by the Anybody authority. With the list of SPs, we now may successively iterate using the Next method over the SPs and, for every one of them, get a list of their tables from the Table table, the same way we did with the SP table. The Table table is again always accessible by the Anybody authority. We then go through the found tables and try to get a list of their rows using the Next method. If this succeeds, we can then use the Get method to get each row of the table. However, some of the tables do not implement the Next method. This means that if we want to get the information from the rows of such tables, we have to guess the UIDs of the rows. Since each table can have up to 2^32 sparsely distributed rows, simply iterating over all possible values is out of the question. In those cases, we only guess that a row with

```
...
# Admin SP
 "0x0000020500000001": {
# Table table
  "0x0000000100000000": {
# Table table row
   "0x0000000100000001": {
# UID
    "0x00": "0x0000000100000001",
# Name
    "0x01": "(Table) 0x5461626c65",
# CommonName
    "0x02": "(Table) 0x5461626c65",
# TemplateID
    "0x03": "0x0000000000000000",
# Kind = Object
    "0x04": "0x01",
...
```

Figure 6.1: Example of `discovery` output from Kingston KC600

the lowest possible UID might exist and try to read information from it.

Even after finishing the crawling, there might still be some tables left that we have not found. Even though the Table table should identify the tables of the SP, some of the tables are not found in the Table table. To solve this issue, we have a list of tables which are described in the Opal standard [8] but were not found in the Table table of some of our tested disks. We then process these tables separately.

During the process, we print the acquired information. We use the JSON format with SPs, tables and rows as dictionaries. The rows then contain the columns as keys and the cells as values. Since the cells can contain diverse types, we have decided to parse only the outermost ones. This means that for example for lists we print the inner bytes in hexadecimal form surrounded by square brackets.

An example of part of the output of this process can be seen in Figure 6.1. Depicted there is a part of the output of this utility, con-

maybe use better example? where the common anme is not used

taining the start of the Table table first row. The nested structure with outermost key corresponding to the Admin SP, followed by keys for the Table table and the row of the Table table for the Table table, can be seen.

wtf, maybe just choose a different row....., are the comments I added better?, add some text that te comments are not part of the output though...

7 Data collection

8 Data (and our utility), deprecated, going to move it into the data collection chapter

maybe one chapter our tool and one chapter the data?

We have implemented a tool to control Opal drives. Compared to the existing solutions, our tool is focused on simplicity and does not require any non-standard library. It is also written in C and therefore does not require any extra runtime dependencies. Our tool is also able to scrape information about the disk, not only limited to Level 0 and Level 1 Discovery introduced in section 3.3, but also Level 2 Discovery, collecting information from all discoverable tables.

In this chapter, we will first describe our tool, its architecture, and its abilities. Afterwards, we will focus on data collected using this tool and its analysis. The source code of the tool can be found in the appendix under the name TODO, and the collected data can be found in the appending under the name ALSOTODO.

8.1 Program

8.1.1 Program structure

Is there something to really talk about?

As the Core standard introduces an entirely? new protocol for the communication and the stream encoding? we had to implement that ourselves.

In order to

8.2 Data

In our analysis we have tested several internal SSD devices, by multiple manufacturers and with different interfaces.

8.2.1 Disk capabilities discovery

In order to collect information about Opal capabilities of the disks, we have written an utility program. This program uses the direct communication described in section 1 to communicate with the disk.

We have decided to use the direct communication instead of the primarily because the capability to perform the discovery process using the Linux Opal ioctl is currently not possible. Even though there are patches being suggested, even if they were to be accepted, only the newest version of Linux kernel would support this feature, limiting the sample size. However, because this approach requires separate implementations of the disk interface commands, something about how this limits us only to a few disk interfaces because of testing, but it's not such a problem because they have a majority share, so find some source that this is actually true. The utility program performs the level 0 and level 1 discovery and the identify command to gather information about the disk. The aggregated and formatted output can be seen in table **??**.

Compared to the two previous levels, the level 2 discovery is more complicated. The level 2 discovery is based on reading tables. Since the standard leaves space for vendor unique tables, we need to first discover these tables. This is done by first reading the SP table of the Admin SP, getting a list of all the SPs in the TPer. Afterwards, we read the Table table of each of the found SPs, getting a list of all tables in each of the SPs. Since each table used in this table discovery process is mandatory, we should be able to get a list of all the tables in the TPer this way. Finally, we can iterate through all the tables, read each entry, and save the contents.

Output of the level 2 discovery depends on several factors: the state of the TPer, the authority used during the discovery... The state of the TPer defines which tables and SPs are available..., It is not desirable to change this state by activating the Locking SP, since that could have unwanted consequences for the future use of Opal on this device by the owner. The admin authority on the other hand could be optionally provided by the user. However, none of the authorities can access all the tables[1], and the information accessible by higher authorities is only potentially sensitive information. So this is a non-issue.

The first part of the table contains values acquired through the level 0 discovery, the second part (starting with `MaxComPacketSize`) con-

1. Counterintuitively, access protected by `ACE_ANYBODY`, can be only be accessed by the Anybody authority and not any of the higher authority authorities, such as Admin authorities.

tains values obtained through the level 2 discovery. In case the disk did not report a value, empty cell is used. There are some noticeable differences: since the first part is reported through firmly established C headers with static form, the numbers are parsed directly as C integers... the second part uses the TCG Storage protocol, with tokens that can be of different sizes — notably in `/dev/sdb` which returns numbers encoded with 4 bytes, even when not necessary. Values of some variables are dependent on each other. For example, because `MaxPackets` is always the minimum (1), then `MaxPacketSize` = `MaxComPacketSize` − (fixed size of `MaxComPacketSize` header).

Although not a part of the discovery process, the utility can also access other functions that tell us more about the device, such as the Random method to generate random data. This provides with potential data to find obviously poorly implemented RNG, as taking larger sample of data for better analysis would take considerable amount of time. TODO: add the average times for getting nice amount of random bytes

To provide more information about the disk, we also perform identify command. This means the IDENTIFY DEVICE command for ATA [3] and the Identify command for NVMe [21].

The data is generated in JSON format. Resulting table transformed into CSV can be seen in the attachment something.

Due to the fact that we use the discovery tools without changing the properties of disk that they are used on (such as by activating the disks), we might not collect the same set of properties for all of the disks. Because of the state of activation, they might differ even between two disk with same hardware and firmware.

8.3 Disk management

Other than the Discovery process described earlier, the utility can also be used for managing a SED device. For this application it offers several commands. The commands allow basic interaction with a SED disk, such as taking ownership of the disk, reverting the disk, creating locking ranges, or locking the disk. However, it is possible to easily extend the tool to support also more advanced use cases, something about how it's thanks to its good quality ;).

```
[root@the-machine-with-disks code]# ./opal_util
   discovery /dev/sda 2>/dev/null
{
"Metadata": {
  "Filename": "/dev/sda"
  "Type": "SATA"
},
"Identify": {
  "Serial␣number": "150912400889␣␣␣␣␣␣␣",
  "Firmware␣revision": "X21803",
  "Model␣number": "SanDisk␣SD7UB2Q512G1122␣␣␣␣␣␣␣
     ␣␣␣␣␣␣␣␣␣"
},
"Discovery␣0": {
"TPer␣Feature": {
 "ComID␣Mgmt␣Supported": 0,
 "Streaming␣Supported": 1,
 "Buffer␣Mgmt␣Supported": 0,
 "ACK/NAK␣Supported": 0,
 "Async␣Supported": 0,
 "Sync␣Supported": 1
},
...
      "0x000000090001ff01": {
        "0x00": "0x000000090001ff01",
        "0x02": "(PhysicalDriveOwner)␣0
           x506879736963616c44726976654f776e6572"
           ,
        "0x03": "0x00",
        "0x04": "0x0000000000000000",
        "0x05": "0x01",
        "0x06": "0x00",
        "0x07": "0x00",
        "0x08": "0x00",
        "0x09": "0x01",
        "0x0a": "0x0000000b0001ff01",
        "0x0b": "0x0000000000000000",
        "0x0c": "0x0000000000000000"
      },
...
```


Figure 8.1: Discovered capabilities of a disk

TODO: something we can do better than sedutil????

8.4 TODOs

TODO: dewscribe how different disks accept "bad" input: some disks don't mind empty list in parameter, other disks hate it (even close the session, some just return empty list as response but success); or alignment of packets:

TODO: using Opal before taking ownership — what happens if I just use default password for everything — works just fine :)

TODO: try to lock only reading, how to write/can I write? – can do just fine :)

TODO: why do some disks have maxinstances of base template one??? TODO: missing DataRemovalMechanism table

TODO: activate sets from manufactured-inactive to manufactured, admin sp starts in manufactured, how to use datastore extension of actiavte then? SPObjectUID.Activate[DataStoreTableSizes = list [uintegers]] => [ ]

9 Conclusion

In the thesis we have introduced several approaches to using hardware-encrypted block devices in Linux. Then, we have focused on the Opal standard more deeply, ... We have then compared capabilities of several Opal-compliant disks. And finally we have provided an overview on attacks on such devices and their feasibility.

As can be easily seen from ...

10 TODO

Holding onto papers here: [4], [25], [26], [5], and [23].

Bibliography

1. *Verizon 2022 Data Breach Investigations Report*. 2022. Tech. rep.
2. FUJIMOTO, Aaron; PETERSON, Peter; REIHER, Peter. Comparing the Power of Full Disk Encryption Alternatives. In: *2012 International Green Computing Conference (IGCC)*. 2012, pp. 1–6. Available from DOI: 10.1109/IGCC.2012.6322245.
3. *Information technology - ATA/ATAPI Command Set - 3 (ACS-3)*. 2013. Tech. rep. American National Standard of Accredited Standards Committee INCITS.
4. MEIJER, Carlo; GASTEL, Bernard van. Self-Encrypting Deception: Weaknesses in the Encryption of Solid State Drives. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 72–87. Available from DOI: 10.1109/SP.2019.00088.
5. ALENDAL, Gunnar; KISON, Christian; MODG. got HW crypto? On the (in)security of a Self-Encrypting Drive series. *IACR Cryptol. ePrint Arch.* 2015, vol. 2015, p. 1002.
6. *About self-encrypting drives*. Available also from: https://www.ibm.com/docs/en/psfa/7.2.1?topic=administration-about-self-encrypting-drives.
7. *Qualcomm® Inline Crypto Engine (UFS)*. 2021-10. Tech. rep.
8. *TCG Storage Security Subsystem Class: Opal*. 2022-01. Version 2.02. Standard. Trusted Computing Group. Available also from: https://trustedcomputinggroup.org/resource/storage-work-group-storage-security-subsystem-class-opal.
9. *TCG Storage Architecture Core Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-architecture-core-specification.
10. *TCG Storage Security Subsystem Class: Opalite Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-opalite.

11. *TCG Storage Security Subsystem Class: Pyrite Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-pyrite.

12. *TCG Storage Security Subsystem Class: Ruby Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-ruby-specification/.

13. *TCG Storage Opal SSC Feature Set: Additional DataStore Tables*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-opal-ssc-feature-set-additional-datastore-tables/.

14. *TCG Storage Feature Set: Block SID Authentication Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-feature-set-block-sid-authentication-specification/.

15. *TCG Storage Opal SSC Feature Set: PSK Secure Messaging*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-opal-ssc-feature-set-psk-secure-messaging/.

16. *TCG Storage Opal SSC Feature Set: PSID*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-opal-feature-set-psid/.

17. *TCG Storage Opal SSC Feature Set: Single User Mode*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-opal-ssc-feature-set-single-user-mode/.

18. *TCG Storage Protection Mechanisms for Secrets Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-protection-mechanisms-for-secrets-specification/.

19. *SED OPAL Library*. Tech. rep. Available also from: https://lore.kernel.org/lkml/1482176149-2257-1-git-send-email-scott.bauer@intel.com/.

20. *TCG Storage Interface Interactions Specification (SIIS)*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/storage-work-group-storage-interface-interactions-specification/.

21. *NVM Express® Base Specification*. Tech. rep. Available also from: https://nvmexpress.org/developers/nvme-specification/.

22. *https://www.t10.org/ftp/t10/document.04/04-262r8.pdf*. Tech. rep.

23. BOTEANU, Daniel; FOWLER, Kevvie. Bypassing Self-Encrypting Drives (SED) in Enterprise Environments. In: 2015.

24. MÜLLER, Tilo; LATZO, Tobias; FREILING, Felix C. *Self-Encrypting Disks pose Self-Decrypting Risks: How to break Hardware-based Full Disk Encryption*. Available also from: https://faui1-files.cs.fau.de/filepool/projects/sed/seds-at-risks.pdf.

25. MÜLLER, Tilo; LATZO, Tobias; FREILING, Felix C.; FRIEDRICH-ALEXANDER. Self-Encrypting Disks pose Self-Decrypting Risks How to break Hardware-based Full Disk Encryption. In: 2013.

26. MÜLLER, Tilo; FREILING, Felix C. A Systematic Assessment of the Security of Full Disk Encryption. *IEEE Transactions on Dependable and Secure Computing*. 2015, vol. 12, no. 5, pp. 491–503. Available from DOI: 10.1109/TDSC.2014.2369041.

27. MÜLLER, Tilo; LATZO, Tobias; FREILING, Felix C.; FRIEDRICH-ALEXANDER. Self-Encrypting Disks pose Self-Decrypting Risks How to break Hardware-based Full Disk Encryption. In: 2013.