MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Hardware-encrypted disks in Linux

Master's Thesis

ŠTĚPÁN HORÁČEK

Brno, Spring 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, Spring 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

This is the abstract of my thesis, which can span multiple paragraphs.

Keywords

keyword1, keyword2, ...

Contents

1 Introduction

Just keeping the citations here, like [1], [2], [3].

1.1 Thesis description

The thesis aims to analyze existing approaches to using hardware-encrypted block devices (disks) in Linux and propose integrating such devices into the cryptsetup project.

The implementation should use the Linux kernel interface.

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 extension to cryptsetup project (and possibly propose changes for Linux kernel).

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

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.

Some general overview about what it is, I guess., maybe talk about both sw and hw enc instead

Describe generic stuff: provisioning, locking, key types DEK, KEK, MEK, processes, locking ranges vs. FDE, ... keyslots?,,,, (and for opal TPer, SP, ..)

mention LUKS somewhere...

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

2.1 Self-encrypting drives

at least try to mention ATA security,,, which means also mentioning opal,,,, and TCG Enterprise...

2.2 Inline encryption hardware

Compared to the previously described self-encrypting drives, inline encryption hardware is separated from the disk, and

something about how it provides actually a way to check the encrypted content, right?

2.3 Software encryption

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

rethink where to put this... maybe tools?

dm-crypt

fscrypt

3 fscrypt

Assuming ext4 in this chapter...

3.1 How to make it work?

As of now, blk-layer inline encryption is supported only by two filesystems in Linux: ext4 and F2FS. Need to use a kernel with CONFIG_FS_ENCRYPTION_INLINE_CRY enabled. Need to first mount the filesystem with the inline encryption flag:

```
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. Assuming ext4 filesystem, the encryption can be enabled after mounting like so:

```
tune2fs -O encrypt "/dev/loop0"
```

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

In order to encrypt a folder using a fscrypt, the following must be done: an encryption key must be added and the encryption policy must be created.

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

struct fscrypt_add_key_arg *key_request = calloc
    (1, sizeof(struct fscrypt_add_key_arg) +
    key_len);
struct fscrypt_policy_v2 policy_request = { 0 };

// add a key
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
policy_request.version = 2;
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.

3.2 How does it work?

3.2.1 Setup

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`.

3.2.2 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 TCG Opal 2.0

TCG Opal SSC (Security Subsystem Class) 2.0 is a specification for storage devices, aiming to provide confidentiality of stored data while the disk implementing this specification is turned off [1]. The Opal standard is one of the representants of the self-encrypting drive approach to hardware disk encryption.

developed by TCG (Trusted Computing Group)

maybe move under the SED chapter, but it is probably going to be too big for that..., and important

structure of the standards described in `https://trustedcomputinggroup.org/wp-content/uploads/TCGandNVMe_Joint_White_Paper-TCG_Storage_Opal_and_NVMe_FINAL.pdf`.

4.1 Technical stuff

I expect stuff here like important headers, codes, ...

SP = service provider, tables and methods that operate upon them

4.1.1 Capability discovery

begin with Level 0 discovery: (4, p. 3.3.6) specifies the general security receives command for all TCG storage devices [1, p. 3.1.1] specifies the Opal 2 specific parts of the header.

The Opal 2 feature header contains several important information usable in the future usage of the device. Most importantly the base ComID [4, p. 3.3.2]. The base ComID, together with another parameter the number of ComIDs, define the range of possible static ComIDs.

ComID are divided into static and dynamic (managed by ComID management).

4.1.2 sessions

In order to facilitate communication between the host and SP, sessions must be used. During these sessions, methods are used...

example of method:

```
SMUID.StartSession [
HostSessionID : uinteger,
SPID : uidref {SPObjectUID},
Write : boolean,
HostChallenge = bytes,
HostExchangeAuthority = uidref {
   AuthorityObjectUID},
HostExchangeCert = bytes,
HostSigningAuthority = uidref {
   AuthorityObjectUID},
HostSigningCert = bytes,
SessionTimeout = uinteger,
TransTimeout = uinteger,
InitialCredit = uinteger,
SignedHash = bytes ]
=>
SMUID.SyncSession [ see SyncSession definition
   in 5.2.3.2]
```

Each method call is defined by the caller UID, method UID, and values of its parameters. The method calls are coded using tokens... start list, end list, optional argument start, ... short int, long int,....

To call a method use IF-SEND command, to get result of the method call use IF-RECV command.

In order to authentize the user, one can use many different ...

Methods are sent using packets. There are three types of packets: ComPackets, packets and subpackets. A single ComPacket can contain multiple packets and single packet can contain multiple subpackets. Each ComPacket is associated with a single ComID, each packet is associated with a single session.

4.1.3 setup

4.1.4 set locking range

global locking range is a special locking range that protects LBAs not protected by any of the other locking ranges.

4.1.5 access...

4.1.6 ...

4.2 Comparison with OPAL 1.0

OPAL 2 disks are *optionally* backwards compatible.

cite `https://trustedcomputinggroup.org/wp-content/uploads/TCG_Storage-Opal_SSC_FAQ.pdf`

4.3 Comparision with other TCG's SSC

Enterprise, Opalite, Pyrite, Ruby

5 Existing tools

5.1 Cryptsetup

... tool for disk encryption setup.

nothing for opal, right?

5.2 sedutil

supports opal, but the code seems to be abandoned

6 Security of hardware encryption

specify threat model, ,....„, offline, online„, offline multiple times???s

.. maybe change position of this chapter, since it's splitting tools and change to a tool

6.1 Attacks on hardware encryption

7 OPAL extension for cryptsetup

8 Conclusion

9 TCG Opal 2.0

TCG Opal Security Subsystem Class (SSC) 2.0 (hereinafter referred to simply as “Opal”) is a specification for storage devices, aiming to provide confidentiality of stored data while the conforming disk is turned off [1]. It is one of the representatives of the self-encrypting drive approach to hardware disk encryption.

Developed in 2012 by Trusted Computing Group (TCG) as a successor of Opal 1.0.

9.1 Structure of the standard

The Opal 2.0 standard is defined as a subsystem extending the TCG Storage Architecture Core Specification standard. The TCG Storage Architecture Core Specification standard [4] specifies the core features and properties shared among several different types of storage security subsystems, 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. 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 [5].

- Pyrite — encryption-less Opalite. Similar to Opalite, however it does not mandate encryption, and instead may offer only logical access control [6].

- Ruby — focused on data centers and server drives. Offers only global range encryption, weaker configuration of access control, no pre-boot authentication support [7]. Replacing the older Enterprise subsystem.

9.2 Architecture

The standard defines several parts of the trusted device.

9.2.1 Trusted Peripheral

Trusted peripheral (TPer) is a device located on the disk that provides the security of the data on disk. A TPer consists of multiple Security Providers.

9.2.2 Security Provider

Security Provider (SP) is defined as a set of tables, methods and an access control. Each SP is derived from a set of templates. These templates define a set of the tables and methods, aimed at one functionality, subset of which the SP implements. The templates are described more closely in later chapter **??**. The Opal 2.0 standard defines that at least the Admin SP and Locking SP must be present in the TPer. The Admin SP 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 chapter **??**.

9.3 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 access the information.

9.3.1 Level 0 discovery

Level 0 discovery provides basic information about the secure device, and is performed using only the secure send and secure receive commands of the device. The information is provided through feature descriptor. The presence of the feature descriptor header means that the feature is supported and the fields of the header describe the basic properties of that feature.

Some of the features described by these descriptors are the TPer feature (supported communication features such as ack/nack support, comid managment, buffer managment, async communication, ...), Locking (whether disk is locked, etc.), Geometry feature (parameters of the disk, such as block size), Opal V1.0 feature, SingleUser feature (...), DataStore (size of the table), OPAL 2.0 feature (base ComID, number of ComIDs, default pin, number of locking users/admins).

9.3.2 Level 1 discovery

Level 1 discovery provides more thorough information about the TPer, and is performed using the `Properties` control session method.

The reported properties are: `MaxMethods`, `MaxSubpackets`, `MaxPacketSize`, `MaxPackets`, `MaxComPacketSize`, `MaxResponseComPacketSize`, `MaxSessions`, `MaxReadSessions`, `MaxIndTokenSize`, `MaxAggTokenSize`, `MaxAuthentications`, `MaxTransactionLimit`, `DefSessionTimeout`, `MaxSessionTimeout`, `MinSessionTimeout`, `DefTransTimeout`, `MaxTransTimeout`, `MinTransTimeout`, `MaxComIDTime`, `ContinuedTokens`, `SequenceNumbers`, `AckNak`, `Asynchronous`,

9.3.3 Level 2 discovery

provided by the `Get` method of SP. This includes reading any table, such as the access control table or table of locking ranges. Some of the SPs' tables are described in later chapter **??**.

9.4 Communication

In order to send commands to the TPer or the SPs and receive responses, a specific protocol must be used. The protocol is split into several layers. The high level method calls, optional transactions, and sessions, which are carried by the interface commands. ((It's actually session, management, communication, TPer, interface, transport, in different part, fixup TODO.))

One of the information required in order to send a 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. This number can be acquired through the level 0 discovery process.

The communication is based on packets. ComPacket is the primary 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 contains one or more method calls, or results of method calls.

9.4.1 Method calls

The data of the SubPacket carries either the method call or method response. These are expressed using a set of tokens in a specific order.

structure is:

```
 CALL_TOKEN | OBJECT_UID | METHOD_UID
| START_LIST_TOKEN | ARGUMENTS | END_LIST_TOKEN | END_OF_DATA_TOKEN
| START_LIST_TOKEN | ERROR_VALUE | END_LIST_TOKEN |
```

the response is identical to the call and different arguments are used as a way of passing back information

object uid can be SP, table, row of table of a special object uid SMUID — session manager, used for session management

mandatory arguments:

optional arguments:

```
 START_NAME_TOKEN | ARGUMENT_ID | ARGUMENT_VALUE | END_NAME_TOKEN
```

9.4.2 Sessions

Sessions can be possibly intertwined even on one ComID

Sessions are using readers-writer lock.

sessions: regular and control (not going to care about control much, just between TPer session manager and host session manager); read/read-write mutexes...,

session manager methods - properties, start/sync session, start/sync trusted session, close session before session one can use the properties method to find a common ground for the capabilities of both host and the Opal device. Some of the values that are agreed upon are the maximum packet size, ...

*trustedsession - used with PuK, SymK, and HMAC authorities, secure messaging

9.4.3 Transactions

In order to facilitate safe execution of sequences of methods, the standard specifies transactions. Similarly to transactions in database systems, this feature enables one to revert effects of 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.

9.4.4 Optional features

Note that some of the features described in the previous sections are optional. Even though every Opal device supports sessions, the minimum required maximum number of sessions is 1, ...

From our experience with Opal disks, most of them implemented only the required minimum <TODO: fact check later on>.

Note that even though in the previous sections it is said that there can be multiple methods per subpacket, or ... these features are all optional. Opal mandates only at least 1 sessions, 1 method per packet, 1 transaction, 1 packet per compacket, 1 subpacket per packet, and so on.

Even though the protocol specifies and can support many features such as secure messaging, asynchronous communication, or session reliability, these features are not mandatory for the Opal subsystem.

9.5 Features

move probably under templates,,, or even better SP?

9.5.1 Single user mode

feature set that "locks" the admin out — admin can do only destructive actions upon the locking range, and only the user can actually unlock it

9.5.2 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 saved shadow MBR on bootup. The shadow MBR may contain a software to authenticate the user to the disk and unlock it. After the shadow MBR is used, it can be deactivated by setting MBR done in the MBRControl table.

The minimal maximum size of the shadow MBR seems to be 0x08000000 bytes (∼134 MB).

Move to locking SP.

9.5.3 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 a 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 credential table. Each SP may contain several credential tables, one for each type of credential, such as pin, RSA key pair, or AES key.

In order to support authentication of more than one user or require multiple users at once access control rules are defined using Access Control Lists (ACL) containing Access Control Elements (ACE). Each entry in an ACL has an owner . . . . ACEs are Boolean expressions with inputs being authentication of authorities.

Other than an explicit authentication, an implicit authentication is also possible. In this authentication, the knowledge of secret is shown by successfully using encrypted communication channel.

authority — object in authority table, type is individual or class (afaik just a bunch of individual authorities,, so that they may be easily changed at once, OR of all the individuals)

9.6 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.

9.6.1 Base template

The 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.

9.6.2 Admin template

specific for admin SP

9.6.3 Crypto template

defines cryptographic methods

9.6.4 Locking template

specific for the locking SP

9.7 Admin SP

holds information about the TPer and SPs, allows modification/creation/deletion of other SPs

unique

9.7.1 Methods

9.8 Locking SP

Locking SP procures the disk encryption and the locking and unlocking associated with it. This means that it provides access to manipulation of locking ranges, key generation and ...

unique

9.8.1 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 has also it's own ACE, to control who can lock and unlock the range, and it's own DEK.

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

9.9 Implementations/Direct usage/something like this

9.9.1 Linux ioctl

Since version 4.11 Linux offers an ioctl to facilitate control over an OPAL disk. Although these ioctl commands offer a simple access to control of the disk, not every feature is implemented, ...

offer only basic commands:

- `SAVE` — something with suspend...
- `LOCK_UNLOCK` — locks/unlocks a locking range
- `TAKE_OWNERSHIP` — initialisation of a TPer, sets up pins
- `ACTIVATE_LSP` —change state from "Manufactured-Inactive" to "Manufactured" state.
- `SET_PW` — change pw
- `ACTIVATE_USR` — sets up an user (change password without old password)
- `REVERT_TPR` — reverts TPer to manufactured state
- `LR_SETUP` — sets locking range position/locking enable
- `ADD_USR_TO_LR` — adds a user to the LR ACE (TODO: they seem to set it to "`(user_uid || user_uid)`"??? why? bug?)
- `ENABLE_DISABLE_MBR` —

- ERASE_LR — calls the erase method, 00 00 06 00 00 08 03, can't find it in opal and in core it's "reserved for SSC"
- SECURE_ERASE_LR — regenerate the key to destroy the previous one
- PSID_REVERT_TPR — resets the TPer using PSID
- MBR_DONE —
- WRITE_SHADOW_MBR —
- GENERIC_TABLE_RW — read/write from/to byte table. no structured table, or iteration based table...

no documentation

currently limited to 1 admin, 9 users (admin is "user0"), no "write only" lock, only basic binary table read (e.g. no iterations, no structured tables)

9.9.2 Direct communication

Alternative to the OPAL ioctl are disk controller ioctls. Depending on the disk protocol, a different way of passing the OPAL commands is required, such as using SG_IO ioctl and sg_io_hdr_t structure for SCSI disks or NVME_IOCTL_ADMIN_CMD ioctl and nvme_admin_cmd structure for NVMe disks. Using these structures, the OPAL commands described in chapter **??** can be sent to the TPer. Compared to the OPAL ioctl this has the advantage of not being limited only to a subset of features that the OPAL ioctl implements, and instead being able to use every OPAL feature the device offers. This is not limited only to . . . (e.g. improve performance by using optional features such as concurency, etc, described earlier) . . .

Although this gives the user access to every feature of the OPAL disk, it also requires them not only to implement the command handover for each type of disk separately, but also to create the methods and parse the method results, both described in chapter **??**, on their own.

10 Notes

Write/read on locked range returns "Input/output error".

tested disks:

- SanDisk SD7UB2Q5
- Samsung SSD 850
  - problems with user authentication

ATA drives need `libata.allow_tpm` set to 1,,, but why?

templates — base, locking,, -> SPs. Service providers of the trusted peripherals are issued based on templates.

Global locking range is a special locking range that controls any area of the disk that is not controlled by any other locking range.

Shadow MBR allows FDE disk pre-boot. Presents “fake” MBR that can unlock the disk, on successful unlocking hands the control over to the real MBR.

key types (Credential Table Group):

- `C_PIN` — password
- `C_RSA_*` — signing arbitrary input???, session startup
- `C_EC_*` — ec-mqv, ec-dh session startup
- `C_AES_*`
- `C_HMAC_*`

5.3.4.1.3

authority operations:

- None — does not authenticate
- Password — authenticated using a pin
- Sign — challenge and response
- Exchange
- SymK

- HMAC
- TPerSign
- TPerExchange

Sessions may use secure messaging: use additionaly a start/synctrustedsession methods for challange-response/key exchange

CVE-2018-12037 — “An issue was discovered on Samsung 840 EVO and 850 EVO devices (only in "ATA high" mode, not vulnerable in "TCG" or "ATA max" mode), …” — wtf is ata high/ata max mode???

CVE-2018-12038

s
e
p
a
r
a
t
o
r

All that follows is from the core standard [4].

secure messaging — integrity/authentication and/or confidentiality,,,,

"Data confidentiality and access control over TPer features and capabilities: . . . The protection provided by this exclusive access extends to confidentiality of instructions and data in transit between the trusted host application (or a TPM it uses) and the TPer"

secure messaging

sessions are using readers-writers lock.

SPs are combinations of templates, must contain base template (). other templates admin, clock, crypto, locking, log

"All SPs incorporate at least a subset of the Base Template's tables and methods." -> is ∅ enough???

issuance = new SP from templates, personalization = customization of newly created SP (but can happen anytime...), initial data, authorities,

SSC = ... "A TPer MAY have only some of the capabilities (tables, methods, access controls, etc.) defined in this Core Specification and MAY include additional capabilities through table definitions and/or methods. A Security Subsystem Class SHALL NOT replace a capability called out in the Core Specification with the same capability implemented in different tables, methods, and access controls."

stream encoding - methods and tokens...,TLV...

tables – bytes tables – raw data, just bytes (rows addressed by row number); object tables (rows addressed by uid, SP-unique, non-reusable, for anti-spoofing)

object = row of table

SP templates – base, admin, clock, crypto, locking, log,

interface – protocol-independent, IF-SEND, IF-RECV commands

COMID

ComID identifies the caller, each application has different ComID and therefore can communicate simultaneously, dynamic, static. Not a session, multiple sessions may use single comid. Then there are some states and transitions, active, associated, issued... Extended ComID for reusing ComID so that we can see if it is the old ComID or the new one (using extra two bytes).

PROTOCOL LAYERS

protocol layers — session, management, communication, tper, interface, transport (... the standard seems to imply that the comid is always managed??)

DISCOVERY

discovery levels : 3.3.5 : level 0 , level 1 Properties method of TPer, level 2 of SP

SESSIONS

sessions: regular and control (not going to care about control much, just between TPer session manager and host session manager); read/read-write mutexes...,

session manager methods - properties, start/sync session, start/sync trusted session, close session

*trustedsession - used with PuK, SymK, and HMAC authorities, secure messaging

authorities – used during session startup,,, host exchange authority – exchange of session keys, implicit authentication;;; host signing authority – challenge response authentication/startup method integrity, `C_PIN` (password), , ,,,, same for host->SP

METHODS

TRANSACTIONS

— like in database, commiting , etc.... optional (or implicit transaction on method level...), nested transaction commited when outermost finished, possible exceptions to rollbacks e.g. logs

Stream Flow Control – interface (handles sending IF-SEND/RECV commands across the interface) and stream data (handles not overwhelming host and TPer) types;;; uses Credit Control Subpacket to signify the opposite device that it is ready to receive data (and how much)

Session Reliability — ACKs, NACKs (SeqNumbers), timeouts,,, all optional

Synchronous Interface Communications — alternative to the previous asynchronous communication... to make it more simple,,,, so IF-SEND to make TPer do something, IF-RECV to get the result, repeat ad nauseam

SP OPERATIONS

Special SP - admin SP – maintains info about TPer and other SPs

SP – Cartesian product of template (defining tables and methods, pretty sure I wrote about it earlier, but it is mentioned in the standard again) subsets,

ACCESS CONTROL

invoker may have to know secret (secret + public part = Credentials, operation of proving knowledge = Authentication Operation, proving knowledge = Authentication)

explicit authentication - e.g. password validation, challenge response„, implicit authentication - e.g session key exchange

10.1 SP Issuance

SP issuance is the process of creation of a new SP from a base template and a set of other templates. After an SP is issued, it can be personalized. Personalization of an SP is the process of

ISSUANCE

issuance – creation of SP from template, from admin SP, „, personalization — after issuance, using the adminexch authority of the new sp, fill in tables, set access control... etc

LIFE CYCLE

... SP can be disabled (still can authenticate, deleteSP and set get on SPinfo (to enable)), frozen (any attempt to open session fails)

s
e
p
a
r
a
t
o
r

... And that§s the introduction... what follows now is the method and table overview...

.....

10.1.1 Templates

The standard specifies several templates that can be used to create SPs.

Base Template –

Crypto Template —- operates on Credential tables (the previously mentioned `C_*`, tables of other SPs,,,), can do enc, dec, sig, ver, hash, hmac, xor

Locking Template —— provides access control

Bibliography

1. *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`.
2. BIGGERS, Eric; TANGIRALA, Satya. *Inline Encryption*. 2021. Available also from: `https://kernel.org/doc/html/v5.17/block/inline-encryption.html`.
3. 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`.
4. *TCG Storage Architecture Core Specification*. Tech. rep. Available also from: `https://trustedcomputinggroup.org/resource/tcg-storage-architecture-core-specification`.
5. *TCG Storage Security Subsystem Class: Opalite Specification*. Tech. rep. Available also from: `https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-opalite`.
6. *TCG Storage Security Subsystem Class: Pyrite Specification*. Tech. rep. Available also from: `https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-pyrite`.
7. *TCG Storage Security Subsystem Class: Ruby Specification*. Tech. rep. Available also from: `https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-ruby-specification/`.