NAME

cryptsetup − utility for configuring and managing encrypted storage devices

SYNOPSIS

cryptsetup <action> [<options>] <action args>

DESCRIPTION

Cryptsetup is a utility for configuring and managing full−disk encryption on storage devices. It can encrypt block devices (such as hard drives or partitions) and containers (disk images stored as files).

When you unlock an encrypted volume, **cryptsetup** creates a new device mapping that applications can access like any regular storage device. The actual encryption and decryption work is performed transparently by the kernel's device−mapper dm−crypt driver.

Cryptsetup works with two main volume types: plain encrypted volumes and LUKS (Linux Unified Key Setup) volumes. Plain volumes provide basic encryption, while LUKS volumes include a metadata header that enables advanced features like multiple keyslots and key management. Additionally, LUKS can be used to manage hardware−based encryption on OPAL−capable storage drives.

Cryptsetup also provides limited support for volumes created by other encryption systems, including **loop−AES**, **TrueCrypt**, **VeraCrypt**, **BitLocker**, and **FileVault2**.

For more information about a specific cryptsetup action, see **cryptsetup−<action>**(8), where **<action>** is the name of the cryptsetup action.

Cryptsetup devices can be activated during boot through **crypttab**(5), which is part of **systemd**(1) or other system init scripts.

BASIC ACTIONS

The following are valid actions for all supported device types.

OPEN

open <device> <name> −−type <device_type>

Opens (creates a mapping with) <name> backed by device <device>. See **cryptsetup−open**(8).

CLOSE

close <name>

Removes the existing mapping <name> and wipes the key from kernel memory. See **cryptsetup−close**(8).

STATUS

status <name>

Reports the status for the mapping <name>. See **cryptsetup−status**(8).

RESIZE

resize <name>

Resizes an active mapping <name>. See **cryptsetup−resize**(8).

REFRESH

refresh <name>

Refreshes parameters of active mapping <name>. See **cryptsetup−refresh**(8).

REENCRYPT

reencrypt <device> or −−active−name <name> [<new_name>]

Run LUKS device reencryption. See **cryptsetup−reencrypt**(8).

PLAIN MODE

Plain dm−crypt encrypts the device sector−by−sector with a single, non−salted hash of the passphrase. No checks are performed, and no metadata is used. There is no formatting operation. When the raw device is mapped (opened), the usual device operations can be used on the mapped device, including filesystem creation. Mapped devices usually reside in /dev/mapper/<name>.

The following are valid plain device type actions:

OPEN

open −−type plain <device> <name>
create <name> <device> (**OBSOLETE syntax**)

Opens (creates a mapping with) <name> backed by device <device>. See **cryptsetup−open**(8).

LUKS EXTENSION

LUKS, the Linux Unified Key Setup, is a standard for disk encryption. It adds a standardized header at the start of the device, a keyslot area directly behind the header and the bulk data area behind that. The whole set is called a 'LUKS container'. The device that a LUKS container resides on is called a 'LUKS device'. For most purposes, both terms can be used interchangeably.

LUKS can manage multiple passphrases that can be individually revoked or changed. Each passphrase uses an individual keyslot containing a volume key for data encryption. Keyslots can be securely scrubbed from persistent media due to the use of anti−forensic stripes. Passphrases are protected against brute−force attacks by the Password−Based Key Derivation Function (PBKDF). A passphrase stored in a file is called a key file. The only difference between a passphrase and a key file is that a key file can contain binary data. Both are processed the same.

LUKS version 1 (or LUKS1) is the original metadata format, while LUKS2 is a new version that allows additional extensions like different PBKDF algorithms or authenticated encryption. You can format the device with a specific LUKS version with **−−type luks1** or **−−type luks2** in the **luksFormat** command. Normally, you do not need to specify any version as it is recognized automatically. The default format is LUKS2.

The **<device>** parameter can also be specified by a LUKS UUID in the format UUID=<uuid>.

The LUKS header can be detached from data (stored separately). To specify a detached header, the −−header parameter can be used in all LUKS commands and always takes precedence over the positional **<device>** parameter.

The following are valid LUKS actions:

FORMAT

luksFormat <device> [<key file>]

Initializes a LUKS partition and sets the initial passphrase (for keyslot 0). See **cryptsetup−luksFormat**(8).

OPEN

open −−type luks <device> <name>
luksOpen <device> <name> (**old syntax**)

Opens the LUKS device <device> and sets up a mapping <name> after successful verification of the supplied passphrase. See **cryptsetup−open**(8).

SUSPEND

luksSuspend <name>

Suspends an active device (all IO operations will block and accesses to the device will wait indefinitely)

and wipes the encryption key from kernel memory. See **cryptsetup−luksSuspend**(8).

RESUME

luksResume <name>

Resumes a suspended device and reinstates the encryption key. See **cryptsetup−luksResume**(8).

ADD KEY

luksAddKey <device> [<key file with new key>]

Adds a new passphrase using an existing passphrase. See **cryptsetup−luksAddKey**(8).

REMOVE KEY

luksRemoveKey <device> [<key file with passphrase to be removed>]

Removes the supplied passphrase from the LUKS device. See **cryptsetup−luksRemoveKey**(8).

CHANGE KEY

luksChangeKey <device> [<new key file>]

Changes an existing passphrase. See **cryptsetup−luksChangeKey**(8).

CONVERT KEY

luksConvertKey <device>

Converts an existing LUKS2 keyslot to new PBKDF parameters. See **cryptsetup−luksConvertKey**(8).

KILL SLOT

luksKillSlot <device> <number>

Wipe the keyslot with the <number> from the LUKS device. See **cryptsetup−luksKillSlot**(8).

ERASE

erase <device>
luksErase <device> (**old syntax**)

Erase all keyslots and make the LUKS container permanently inaccessible. See **cryptsetup−erase**(8).

UUID

luksUUID <device>

Print or set the UUID of a LUKS device. See **cryptsetup−luksUUID**(8).

IS LUKS

isLuks <device>

Returns true, if <device> is a LUKS device, false otherwise. See **cryptsetup−isLuks**(8).

DUMP

luksDump <device>

Dump the header information of a LUKS device. See **cryptsetup−luksDump**(8).

HEADER BACKUP

luksHeaderBackup <device> −−header−backup−file <file>

Stores a binary backup of the LUKS header and keyslot area. See **cryptsetup−luksHeaderBackup**(8).

HEADER RESTORE

luksHeaderRestore <device> −−header−backup−file <file>

Restores a binary backup of the LUKS header and keyslot area from the specified file. See

cryptsetup−luksHeaderRestore(8).

TOKEN

token <add|remove|import|export> <device>

Manipulate token objects used for obtaining passphrases. See **cryptsetup−token**(8).

CONVERT

convert <device> −−type <format>

Converts the device between LUKS1 and LUKS2 format (if possible). See **cryptsetup−convert**(8).

CONFIG

config <device>

Set permanent configuration options (store to LUKS header). See **cryptsetup−config**(8).

LOOP−AES EXTENSION

Cryptsetup supports mapping a loop−AES encrypted partition using a compatibility mode.

OPEN

open −−type loopaes <device> <name> −−key−file <keyfile>
loopaesOpen <device> <name> −−key−file <keyfile> (**old syntax**)

Opens the loop−AES <device> and sets up a mapping <name>. See **cryptsetup−open**(8).

See also section 7 of the FAQ and loop−AES <http://loop−aes.sourceforge.net> for more information regarding loop−AES.

TCRYPT (TRUECRYPT AND VERACRYPT COMPATIBLE) EXTENSION

Cryptsetup supports mapping of TrueCrypt, tcplay, or VeraCrypt encrypted partitions using a native Linux kernel API. Header formatting and TCRYPT header change are not supported; cryptsetup never changes the TCRYPT header on−device.

TCRYPT extension requires the kernel userspace crypto API to be available. If you are configuring the kernel yourself, enable "User−space interface for symmetric key cipher algorithms" in "Cryptographic API" section (CRYPTO_USER_API_SKCIPHER .config option).

Because the TCRYPT header is encrypted, you must always provide a valid passphrase and keyfiles.

Cryptsetup should recognize all header variants, except legacy cipher chains using LRW encryption mode with a 64−bit encryption block (namely, Blowfish in LRW mode is not recognized; this is a limitation of the kernel crypto API).

VeraCrypt is an extension of TrueCrypt with an increased iteration count, so unlocking can take quite a lot of time.

To open a VeraCrypt device with a custom Personal Iteration Multiplier (PIM) value, use either the −−veracrypt−pim PIM option to directly specify the PIM on the command line or use −−veracrypt−query−pim to be prompted for the PIM.

The PIM value affects the number of iterations applied during key derivation. Please refer to PIM <https://veracrypt.io/en/Personal%20Iterations%20Multiplier%20(PIM).html> for more detailed information.

If you need to disable VeraCrypt device support, use −−disable−veracrypt option.

Activation with **tcryptOpen** is supported only for cipher chains using LRW or XTS encryption modes.

The **tcryptDump** command should work for all recognized TCRYPT devices and doesn't require superuser privilege.

To map the system device (device with boot loader where the whole encrypted system resides), use −−tcrypt−system option. Please read specific info in **cryptsetup−tcryptOpen**(8) −−tcrypt−system option section as mapping system−encrypted device is tricky.

To use a hidden header (and map hidden device, if available), use −−tcrypt−hidden option.

To explicitly use the backup (secondary) header, use −−tcrypt−backup option.

There is no protection for a hidden volume if the outer volume is mounted. The reason is that if there were any protection, it would require some metadata describing what to protect in the outer volume, and the hidden volume would become detectable.

OPEN

open −−type tcrypt <device> <name>
tcryptOpen_ <device> <name> (**old syntax**)

Opens the TCRYPT (a TrueCrypt−compatible) <device> and sets up a mapping <name>. See **cryptsetup−open**(8).

DUMP

tcryptDump <device>

Dump the header information of a TCRYPT device. See **cryptsetup−tcryptDump**(8).

See also TrueCrypt <https://en.wikipedia.org/wiki/TrueCrypt> and VeraCrypt <https://en.wikipedia.org/wiki/VeraCrypt> pages for more information.

Please note that cryptsetup does not use TrueCrypt or VeraCrypt code; please report all problems related to this compatibility extension to the cryptsetup project.

BITLK (WINDOWS BITLOCKER COMPATIBLE) EXTENSION

Cryptsetup supports mapping of BitLocker and BitLocker to Go encrypted partitions using a native Linux kernel API. Header formatting and BITLK header changes are not supported; cryptsetup never changes the BITLK header on−device.

BITLK extension requires the kernel userspace crypto API to be available (for details, see the TCRYPT section).

Cryptsetup should recognize all BITLK header variants, except the legacy header used in Windows Vista systems and partially decrypted BitLocker devices. Activation of legacy devices encrypted in CBC mode requires at least a Linux kernel version 5.3, and for devices using the Elephant diffuser, kernel 5.6.

The **bitlkDump** command should work for all recognized BITLK devices and doesn't require superuser privilege.

For unlocking with the **open**, a password, a recovery passphrase, or a startup key must be provided.

Additionally, unlocking using the volume key is supported. You must provide BitLocker Full Volume Encryption Key (FVEK) using the −−volume−key−file option. The key must be decrypted and without the header (only 128/256/512 bits of key data depending on the used cipher and mode).

Other unlocking methods (TPM, SmartCard) are not supported.

OPEN

open −−type bitlk <device> <name>
bitlkOpen <device> <name> (**old syntax**)

Opens the BITLK (a BitLocker−compatible) <device> and sets up a mapping <name>. See **cryptsetup−open**(8).

DUMP

bitlkDump <device>

Dump the header information of a BITLK device. See **cryptsetup−bitlkDump**(8).

Please note that cryptsetup does not use any Windows BitLocker code; please report all problems related to this compatibility extension to the cryptsetup project.

FVAULT2 (APPLE MACOS FILEVAULT2 COMPATIBLE) EXTENSION

Cryptsetup supports the mapping of FileVault2 (FileVault2 full−disk encryption) by Apple for the macOS operating system using a native Linux kernel API.

Cryptsetup supports only FileVault2 based on Core Storage and HFS+ filesystem (introduced in MacOS X 10.7 Lion). It does NOT support the new version of FileVault based on the APFS filesystem used in recent macOS versions.

Header formatting and FVAULT2 header changes are not supported; cryptsetup never changes the FVAULT2 header on−device.

FVAULT2 extension requires the kernel userspace crypto API to be available (for details, see the TCRYPT section) and a kernel driver for the HFS+ (hfsplus) filesystem.

Cryptsetup should recognize the basic configuration for portable drives.

The **fvault2Dump** command should work for all recognized FVAULT2 devices and doesn't require superuser privilege.

For unlocking with the **open**, a password must be provided. Other unlocking methods are not supported.

OPEN

open −−type fvault2 <device> <name>
fvault2Open <device> <name> (**old syntax**)

Opens the FVAULT2 (a FileVault2−compatible) <device> (usually the second partition on the device) and sets up a mapping <name>. See **cryptsetup−open**(8).

SED (SELF ENCRYPTING DRIVE) OPAL EXTENSION

Cryptsetup supports using native hardware encryption on drives that provide an **OPAL** interface, both nested with **dm−crypt** and standalone. Passphrases, tokens and metadata are stored using the LUKS2 header format, and are thus compatible with any software or system that uses LUKS2 (e.g., tokens).

OPAL support requires at least kernel v6.4. Resizing devices is not supported.

The −−hw−opal can be specified for OPAL + dm−crypt, and −−hw−opal−only can be specified to use OPAL only, without a dm−crypt layer.

Opening, closing and enrolling tokens work the same way as with LUKS2 and dm−crypt. The new parameters are only necessary when formatting; the LUKS2 metadata will ensure the right setup is performed when opening or closing.

If no **subsystem** label is specified, it will be automatically set to **HW−OPAL** so that it is immediately apparent when a device uses OPAL.

FORMAT

luksFormat −−type luks2 −−hw−opal <device> [<key file>]

Additionally specify −−hw−opal−only instead of −−hw−opal to avoid the dm−crypt layer. Other than the usual passphrase, an admin password will have to be specified when formatting the drive's first partition, and will have to be re−supplied when formatting any other partition until a factory reset is performed.

ERASE

erase <device>

Securely erase a partition or device. Requires admin password. Additionally specify −−hw−opal−factory−reset for a FULL factory reset of the drive, using the drive's **PSID** (typically printed on the label) instead of the admin password.

PSID must be entered without dashes, spaces or underscores.

WARNING: A factory reset will cause ALL data on the device to be lost, regardless of the partition it is run on, if any, and regardless of any LUKS2 header backup.

MISCELLANEOUS ACTIONS

REPAIR

repair <device>

Tries to repair the device metadata if possible. Currently supported only for LUKS device type. See **cryptsetup−repair**(8).

BENCHMARK

benchmark <options>

Benchmarks, ciphers and KDF (key derivation function). See **cryptsetup−benchmark**(8).

PLAIN MODE OR LUKS?

Unless you understand the cryptographic background well, use LUKS. With plain mode, there are a number of possible user errors that massively decrease security. While LUKS cannot fix them all, it can lessen the impact for many of them.

WARNINGS

A lot of good information on the risks of using encrypted storage, on handling problems and on security aspects can be found in the Cryptsetup FAQ. Read it. Nonetheless, some risks deserve to be mentioned here.

Backup: Storage media die. Encryption has no influence on that. Backup is mandatory for encrypted data as well, if the data has any worth. See the Cryptsetup FAQ for advice on how to back up an encrypted volume.

Character encoding: If you enter a passphrase with special symbols, the passphrase can change depending on character encoding. Keyboard settings can also be changed, which can make blind input hard or impossible. For example, switching from some ASCII 8−bit variant to UTF−8 can lead to a different binary encoding and hence a different passphrase seen by cryptsetup, even if what you see on the terminal is exactly the same. It is therefore highly recommended to select passphrase characters only from 7−bit ASCII, as the encoding for 7−bit ASCII stays the same for all ASCII variants and UTF−8.

LUKS header: If the header of a LUKS volume gets damaged, all data is permanently lost unless you have a header backup. If a keyslot is damaged, it can only be restored from a header backup or if another active keyslot with a known passphrase is undamaged. This risk is the result of a trade−off between security and

safety, as LUKS is designed for fast and secure wiping by just overwriting the header and keyslot area.

Previously used partitions: If a partition was previously used, it is a very good idea to wipe filesystem signatures, data, etc., before creating a LUKS or plain dm−crypt container. For a quick removal of filesystem signatures, use **wipefs**(8) with the −−all option. Note that it does not remove data; it only invalidates known format signatures. For a full wipe, overwrite the whole partition before creating a container. If you do not know how to do that, the cryptsetup FAQ describes several options.

EXAMPLES

Example 1: Create LUKS 2 container on block device /dev/sdX.

```
sudo cryptsetup −−type luks2 luksFormat /dev/sdX
```

Example 2: Add an additional passphrase to keyslot 5.

```
sudo cryptsetup luksAddKey −−key−slot 5 /dev/sdX
```

Example 3: Create LUKS header backup and save it to a file.

```
sudo cryptsetup luksHeaderBackup /dev/sdX −−header−backup−file /var/tmp/NameOfBackupFile
```

Example 4: Open LUKS container on /dev/sdX and map it to sdX_crypt.

```
sudo cryptsetup open /dev/sdX sdX_crypt
```

WARNING: The command in example 5 will erase all keyslots.

You cannot use your LUKS container afterward anymore unless you have a backup to restore.

Example 5: Erase all keyslots on /dev/sdX.

```
sudo cryptsetup erase /dev/sdX
```

Example 6: Restore LUKS header from backup file.

```
sudo cryptsetup luksHeaderRestore /dev/sdX −−header−backup−file /var/tmp/NameOfBackupFile
```

RETURN CODES

Cryptsetup returns **0** on success and a non−zero value on error.

Error codes are: **1** wrong parameters, **2** no permission (bad passphrase), **3** out of memory, **4** wrong device specified, **5** device already exists or device is busy.

NOTES

Passphrase processing for PLAIN mode

Note that no iterated hashing or salting is done in plain mode. If hashing is done, it is a single direct hash. This means that low−entropy passphrases are easy to attack in plain mode.

From a terminal: The passphrase is read until the first newline, i.e., '\n'. The input without the newline character is processed with the default hash or the hash specified with −−hash. The hash result will be truncated to the key size of the used cipher, or the size specified with −s.

From stdin: Reading will continue until a newline (or until the maximum input size is reached), with the trailing newline stripped. The maximum input size is defined by the same compiled−in default as the maximum key file size and can be overwritten using the −−keyfile−size option.

The data read will be hashed with the default hash or the hash specified with −−hash. The hash result will be truncated to the key size of the used cipher, or the size specified with −s.

Note that if −−key−file=− is used for reading the key from stdin, trailing newlines are not stripped from the input.

If "plain" is used as an argument to −−hash, the input data will not be hashed. Instead, it will be

zero−padded (if shorter than the key size) or truncated (if longer than the key size) and used directly as the binary key. This is useful for directly specifying a binary key. No warning will be given if the amount of data read from stdin is less than the key size.

From a key file: It will be truncated to the key size of the used cipher or the size given by −s and directly used as a binary key.

The −−hash argument is being ignored. The −−hash option is usable only for stdin input in plain mode.

If the key file is shorter than the key, cryptsetup will quit with an error. The maximum input size is defined by the same compiled−in default as the maximum key file size and can be overwritten using the −−keyfile−size option.

Passphrase processing for LUKS

From a terminal: The passphrase is read until the first newline and then processed by PBKDF2 without the newline character.

From stdin: LUKS will read passphrases from stdin up to the first newline character or the compiled−in maximum key file length. If −−keyfile−size is given, it is ignored.

From key file: The complete keyfile is read up to the compiled−in maximum size. Newline characters do not terminate the input. The −−keyfile−size option can be used to limit what is read.

LUKS uses **Password−Based Key Derivation Function** (PBKDF) to protect against brute−force attacks and to give some protection to low−entropy passphrases (see cryptsetup FAQ). LUKS1 supports the PBKDF2 algorithm only, while LUKS2 also supports memory−hard Argon2. PBKDFs are configured with costs: how long the iteration should run (CPU cost or iteration count), how much memory is used (memory cost), and how many parallel processes are used (parallel cost). PBKDF2 supports only iteration count. Cryptsetup uses PBKDF benchmarking to calculate optimal costs based on the computer where the new passphrase is being initialized. If needed, these costs can also be overwritten. Note that there are some hardcoded limits, for details see **MINIMAL AND MAXIMAL PBKDF COSTS** section in −−pbkdf option description.

Whenever a passphrase is added to a LUKS header (luksAddKey, luksFormat), the user may specify how much time the passphrase processing should consume. The time is used to determine the iteration count for PBKDF2, and higher times will offer better protection for low−entropy passphrases, but the open command will take longer to complete. For passphrases that have entropy higher than the used key length, higher iteration times will not increase security.

The default setting of one or two seconds is sufficient for most practical cases. The only exception is a low−entropy passphrase used on a device with a slow CPU, as this will result in a low iteration count. On a slow device, it may be advisable to increase the iteration time using the −−iter−time option to obtain a higher iteration count. This does slow down all later luksOpen operations accordingly.

Incoherent behavior for invalid passphrases/keys

LUKS checks for a valid passphrase when a keyslot is decrypted.

The behavior of plain dm−crypt is different. It will always unlock the device with the passphrase given. If the given passphrase is wrong, the device mapped by plain dm−crypt will use the wrong encryption key, and the data will be unreadable.

Supported ciphers, modes, hashes and key sizes

The available combinations of ciphers, modes, hashes and key sizes depend on kernel support. See /proc/crypto for a list of available options. You might need to load additional kernel crypto modules to get more options.

Cryptsetup processes many operations outside of the kernel, so the configured cryptographic library must also support selected algorithms. Some algorithms may be missing as cryptsetup can be compiled with various cryptographic backends (libraries).

Notes on passphrases

Mathematics can't be bribed. Make sure you keep your passphrases safe. There are a few nice tricks for constructing a fallback when suddenly, out of the blue, your brain refuses to cooperate. These fallbacks need LUKS, as it's only possible with LUKS to have multiple passphrases. Still, if your attacker model does not prevent it, storing your passphrase in a sealed envelope somewhere may be a good idea as well.

Notes on Random Number Generators

Random Number Generators (RNGs) used in cryptsetup are always the kernel RNGs without any modifications or additions to the data stream produced.

There are two types of randomness that cryptsetup/LUKS needs. One type is used for salts, the AF splitter and for wiping deleted keyslots. The second type is used for the volume key.

With recent kernels (Linux kernel 5.6), you do not need to worry about selecting RNG (/dev/random or /dev/urandom). In a low−entropy situation (embedded system), initialization of the kernel RNG can take a very long time, but this happens before cryptsetup can even be started. Use *cryptsetup −−help* to show the compiled−in default random number generator. See **urandom**(4) for more information.

Authenticated disk encryption (EXPERIMENTAL)

Normal disk encryption modes are length−preserving (the plaintext sector is the same size as a ciphertext sector) and can provide only confidentiality protection, not cryptographically sound data integrity protection.

Authenticated modes require additional space per−sector for the authentication tag and use Authenticated Encryption with Additional Data (AEAD) algorithms.

If you configure a LUKS2 device with data integrity protection, there will be an underlying dm−integrity device, which provides additional per−sector metadata space and data journal protection to ensure atomicity of data and metadata updates. Because there must be additional space for metadata and journal, the available space for the device will be smaller than for length−preserving modes.

The dm−crypt device then resides on top of such a dm−integrity device. All activation and deactivation of this device stack is performed by cryptsetup; there is no difference in using **luksOpen** for integrity−protected devices. If you want to format a LUKS2 device with data integrity protection, use −−integrity option (see **cryptsetup−luksFormat**(8)).

Albeit Linux kernel 5.7 added TRIM support for standalone dm−integrity devices, **cryptsetup**(8) can't offer support for discards (TRIM) in authenticated encryption mode, because the underlying dm−crypt kernel module does not support this functionality when dm−integrity is used as auth tag space allocator (see −−allow−discards in **cryptsetup−open**(8)).

Some integrity modes require two independent keys (a key for encryption and authentication). Both these keys are stored in one LUKS keyslot.

Support for authenticated modes is experimental, and only some modes are available now. Note that very few authenticated encryption algorithms are suitable for disk encryption. You also cannot use CRC32 or other non−cryptographic checksums (other than the special integrity mode "none"). If, for some reason, you want to have integrity control without using authentication mode, then you should separately configure dm−integrity independently of LUKS2.

Notes on loopback device use

Cryptsetup is usually used directly on a block device (disk partition or LVM volume). However, if the device argument is a file, cryptsetup tries to allocate a loopback device and map it into this file. Of course,

you can always map a file to a loop device manually. See the cryptsetup FAQ for an example.

When device mapping is active, you can see the loop backing file in the status command output. Also see losetup(8).

LUKS2 header locking

The LUKS2 on−disk metadata is updated in several steps, and to achieve a proper atomic update, there is a locking mechanism. For an image in a file, the code uses the **flock**(2) system call. For a block device, lock is performed over a special file stored in a locking directory (by default **/run/cryptsetup**). The locking directory should be created with the proper security context by the distribution during the boot−up phase. Only LUKS2 uses locks; other formats do not use this mechanism.

LUKS on−disk format specification

For LUKS on−disk metadata specification, see LUKS1 <https://gitlab.com/cryptsetup/cryptsetup/wikis/Specification> and LUKS2 <https://gitlab.com/cryptsetup/LUKS2−docs>.

AUTHORS

Cryptsetup was originally written by Jana Saout <jana@saout.de>. The LUKS extensions and original man page were written by Clemens Fruhwirth <clemens@endorphin.org>. Man page extensions by Milan Broz <gmazyland@gmail.com>. Man page rewrite and extension by Arno Wagner <arno@wagner.name>.

REPORTING BUGS

Report bugs at cryptsetup mailing list <cryptsetup@lists.linux.dev> or in Issues project section <https://gitlab.com/cryptsetup/cryptsetup/−/issues/new>.

Please attach the output of the failed command with −−debug option added.

SEE ALSO

Cryptsetup FAQ <https://gitlab.com/cryptsetup/cryptsetup/wikis/FrequentlyAskedQuestions>

cryptsetup(8), **integritysetup**(8) and **veritysetup**(8)

CRYPTSETUP

Part of cryptsetup project <https://gitlab.com/cryptsetup/cryptsetup/>.