MyWebUniversity

Red Hat Enterprise Linux Manual Pages

FPDF Library PDF generator

Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'cryptsetup.8' command

$ man cryptsetup.8

CRYPTSETUP(8) Maintenance Commands CRYPTSETUP(8)

NAME

cryptsetup - manage plain dm-crypt, LUKS, and other encrypted volumes

SYNOPSIS

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

DESCRIPTION

cryptsetup is used to conveniently setup dm-crypt managed device-mapper mappings. These include plain dm-crypt volumes and LUKS volumes. The difference is that LUKS uses a metadata header and can hence offer more features than plain dm-crypt. On the other hand, the header is visible and vulnerable to damage.

In addition, cryptsetup provides limited support for the use of loop-AES volumes, TrueCrypt, VeraCrypt, BitLocker and FileVault2 compatible volumes.

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

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, 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 key-slot 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. But note that when the LUKS header is at a nonzero offset in a device, then the device is not a LUKS device anymore, but has a LUKS container stored in it at an offset.

LUKS can manage multiple passphrases that can be individually revoked or changed and that can be securely scrubbed from persistent media due to the use of anti-forensic stripes. Passphrases are protected against brute-force and dictionary attacks by Password-Based Key Derivation Function (PBKDF).

LUKS2 is a new version of header format that allows additional extensions like different PBKDF algorithm or authenticated encryption. You can format device with LUKS2 header if you specify --type luks2 in luksFormat command. For activation, the format is already recognized automatically.

Each passphrase, also called a key in this document, is associated with one of 8 key-slots. Key operations that do not specify a slot affect the first slot that matches the supplied passphrase or the first empty slot if a new passphrase is added.

The <device> parameter can also be specified by a LUKS UUID in the format UUID=<uuid>. Translation to real device name uses symlinks in /dev/disk/by-uuid directory.

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

key-slot 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> <key slot number>

Wipe the key-slot number <key slot> 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 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 partition using a native Linux kernel API. Header formatting and TCRYPT header change is not supported, cryptsetup never changes TCRYPT header on-device.

TCRYPT extension requires kernel userspace crypto API to be available (introduced in Linux kernel 2.6.38). If you are configuring kernel yourself, enable "User-space interface for symmetric key cipher algorithms" in "Cryptographic API" section (CRYPTO_USER_API_SKCIPHER .config option).

Because TCRYPT header is encrypted, you have to always provide valid passphrase and keyfiles.

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

VeraCrypt is extension of TrueCrypt header with 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://www.veracrypt.fr/en/Personal%20Iterations%20Multiplier%20%28PIM%29.html> for more detailed information.

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

NOTE: 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 system device (device with boot loader where the whole encrypted system resides) use --tcrypt-system option. You can use partition device as the parameter (parameter must be real partition device, not an image in a file), then only this partition is mapped.

If you have the whole TCRYPT device as a file image and you want to map multiple partition encrypted with system encryption, please create loopback mapping with partitions first (losetup -P, see losetup(8) man page for more info), and use loop partition as the device parameter.

If you use the whole base device as a parameter, one device for the whole system encryption is mapped. This mode is available only for backward compatibility with older cryptsetup versions which mapped

TCRYPT system encryption using the whole device.

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

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

NOTE: 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 partition using a native Linux kernel API. Header formatting and BITLK header changes are not supported, cryptsetup never changes BITLK header on-device.

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

Cryptsetup should recognize all BITLK header variants, except legacy header used in Windows Vista systems and partially decrypted BitLocker devices. Activation of legacy devices encrypted in CBC mode requires at least Linux kernel version 5.3 and for devices using 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 or a recovery passphrase or a startup key must be provided.

Additionally unlocking using 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 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.

NOTE: 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 kernel userspace crypto API to be available

(for details, see TCRYPT section) and kernel driver for 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).

DUMP

fvault2Dump <device>

Dump the header information of an FVAULT2 device.

See cryptsetup-fvault2Dump(8).

Note that cryptsetup does not use any macOS code or proprietary

specifications. Please report all problems related to this

compatibility extension to the cryptsetup project.

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 DM-CRYPT OR LUKS?

Unless you understand the cryptographic background well, use LUKS. With plain dm-crypt 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 do a backup of 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 change, 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 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 key-slot is damaged, it can only be restored from a header-backup or if another active key-slot with known passphrase is undamaged. Damaging the LUKS header is something people manage to do with surprising frequency. 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 header and key-slot 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 on it. For a quick removal of filesystem signatures, use wipefs(8). Take care though that this may not remove everything. In particular, MD RAID signatures at the end of a device may survive. It also does not remove data. For a full wipe, overwrite the whole partition before container creation. 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 key slot 5.

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

Example 3: Create LUKS header backup and save it to 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 key slots.

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

Example 5: Erase all key slots 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 for the maximum key file size and can be overwritten using --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 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.

WARNING: 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 for the maximum key file size and can be overwritten using --keyfile-size option.

LUKS uses PBKDF to protect against dictionary attacks and to give some protection to low-entropy passphrases (see cryptsetup FAQ).

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.

Passphrase processing: Whenever a passphrase is added to a LUKS header (luksAddKey, luksFormat), the user may specify how much the 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 open 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 in order 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 an encrypted partition is unlocked. The behavior of plain dm-crypt is different. It will always decrypt with the passphrase given. If the given passphrase is wrong, the device mapped by plain dm-crypt will essentially still contain encrypted data and 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 in order to get more options.

For the --hash option, if the crypto backend is libgcrypt, then all algorithms supported by the gcrypt library are available. For other crypto backends, some algorithms may be missing.

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

There are two types of randomness cryptsetup/LUKS needs. One type (which always uses /dev/urandom) is used for salts, the AF splitter and for wiping deleted keyslots.

The second type is used for the volume key. You can switch between using /dev/random and /dev/urandom here, see --use-random and --use-urandom options. Using /dev/random on a system without enough entropy sources can cause luksFormat to block until the requested amount of random data is gathered. In a low-entropy situation (embedded system), this can take a very long time and potentially forever. At the same time, using /dev/urandom in a low-entropy situation will produce low-quality keys. This is a serious problem, but solving it is out of scope for a mere man-page. See urandom(4) for more information.

Authenticated disk encryption (EXPERIMENTAL)

Since Linux kernel version 4.12 dm-crypt supports authenticated disk encryption.

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

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

If you configure LUKS2 device with data integrity protection, there will be an underlying dm-integrity device, which provides additional per-sector metadata space and also provide data journal protection to ensure atomicity of data and metadata update. 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 LUKS2 device with data integrity protection, use --integrity option.

Since dm-integrity doesn?t support discards (TRIM), dm-crypt device on top of it inherits this, so integrity protection mode doesn?t support discards either.

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

WARNING: All support for authenticated modes is experimental and there are only some modes available for now. Note that there are a very few authenticated encryption algorithms that are suitable for disk encryption. You also cannot use CRC32 or any 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. This mode requires Linux kernel 2.6.25 or more recent which supports the loop autoclear flag (loop device is cleared on the last close automatically). 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 proper atomic update, there is a locking mechanism. For an image in file, code uses 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 is 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 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)

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