NAME

cryptsetup - manage plain dm-crypt and LUKS encrypted volumes

SYNOPSIS

cryptsetup <options> <action> <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 and BitLocker compatible volumes.

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". 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.

BASIC ACTIONS

The following are valid actions for all supported device types.

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

Opens (creates a mapping with) <name> backed by device <device>.

Device type can be *plain*, *luks* (default), *luks1*, *luks2*, *loopaes* or *tcrypt*.

For backward compatibility there are **open** command aliases:

create (argument-order <name> <device>): open −−type plain
plainOpen: open −−type plain
luksOpen: open −−type luks
loopaesOpen: open −−type loopaes
tcryptOpen: open −−type tcrypt
bitlkOpen: open −−type bitlk

<options> are type specific and are described below for individual device types. For **create**, the order of the <name> and <device> options is inverted for historical reasons, all other aliases use the standard **<device> <name>** order.

close <name>

Removes the existing mapping <name> and wipes the key from kernel memory.

For backward compatibility there are **close** command aliases: **remove**, **plainClose**, **luksClose**, **loopaesClose**, **tcryptClose** (all behaves exactly the same, device type is determined automatically from active device).

<options> can be [−−deferred] or [−−cancel−deferred]

status <name>

Reports the status for the mapping <name>.

resize <name>

Resizes an active mapping <name>.

If −−size (in 512-bytes sectors) or −−device−size are not specified, the size is computed from the underlying device. For LUKS it is the size of the underlying device without the area reserved for LUKS header (see data payload offset in **luksDump** command). For plain crypt device, the whole device size is used.

Note that this does not change the raw device geometry, it just changes how many sectors of the raw device are represented in the mapped device.

If cryptsetup detected volume key for active device loaded in kernel keyring service, resize action would first try to retrieve the key using a token and only if it failed it'd ask for a passphrase to unlock a keyslot (LUKS) or to derive a volume key again (plain mode). The kernel keyring is used by default for LUKS2 devices.

With LUKS2 device additional **<options>** can be [−−token−id, −−token−only, −−token-type, −−key−slot, −−key−file, −−keyfile−size, −−keyfile−offset, −−timeout, −−disable−external−tokens, −−disable−locks, −−disable−keyring].

refresh <name>

Refreshes parameters of active mapping <name>.

Updates parameters of active device <name> without need to deactivate the device (and umount filesystem). Currently it supports parameters refresh on following devices: LUKS1, LUKS2 (including authenticated encryption), plain crypt and loopaes.

Mandatory parameters are identical to those of an open action for respective device type.

You may change following parameters on all devices −−perf−same_cpu_crypt, −−perf−submit_from_crypt_cpus, −−perf-no_read_workqueue, −−perf-no_write_workqueue and −−allow−discards.

Refreshing device without any optional parameter will refresh the device with default setting (respective to device type).

LUKS2 only:

−−integrity−no−journal parameter affects only LUKS2 devices with underlying dm-integrity device.

Adding option −−persistent stores any combination of device parameters above in LUKS2 metadata (only after successful refresh operation).

−−disable−keyring parameter refreshes a device with volume key passed in dm-crypt driver.

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

Run resilient reencryption (LUKS2 device only).

There are 3 basic modes of operation:

• device reencryption (*reencrypt*)

• device encryption (*reencrypt* −−encrypt)

• device decryption (*reencrypt* −−decrypt)

<device> or --active-name <name> is mandatory parameter.

With <device> parameter cryptsetup looks up active <device> dm mapping. If no active mapping is detected, it starts offline reencryption otherwise online reencryption takes place.

Reencryption process may be safely interrupted by a user via SIGTERM signal (ctrl+c).

To resume already initialized or interrupted reencryption, just run the cryptsetup *reencrypt* command again to continue the reencryption operation. Reencryption may be resumed with different −−resilience or −−hotzone−size unless implicit datashift resilience mode is used (reencrypt −−encrypt with −−reduce-device-size option).

If the reencryption process was interrupted abruptly (reencryption process crash, system crash, poweroff) it may require recovery. The recovery is currently run automatically on next activation (action *open*) when needed.

Optional parameter <new_name> takes effect only with −−encrypt option and it activates device <new_name> immediately after encryption initialization gets finished. That's useful when device needs to be ready as soon as possible and mounted (used) before full data area encryption is completed.

Action supports following additional **<options>** [−−encrypt, −−decrypt, −−device−size, −−resilience, −−resilience-hash, −−hotzone-size, −−init−only, −−resume−only, −−reduce−device−size, −−master−key−file, −−key−size].

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

Opens (creates a mapping with) <name> backed by device <device>.

<options> can be [−−hash, −−cipher, −−verify-passphrase, −−sector−size, −−key-file, −−keyfile-offset, −−key-size, −−offset, −−skip, −−size, −−readonly, −−shared, −−allow−discards, −−refresh]

Example: 'cryptsetup open −−type plain /dev/sda10 e1' maps the raw encrypted device /dev/sda10 to the mapped (decrypted) device /dev/mapper/e1, which can then be mounted, fsck-ed or have a filesystem created on it.

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 PBKDF2, which implements hash iteration and salting in one function.

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:

luksFormat <device> [<key file>]

Initializes a LUKS partition and sets the initial passphrase (for key-slot 0), either via prompting or via <key file>. Note that if the second argument is present, then the passphrase is taken from the file given there, without the need to use the −−key-file option. Also note that for both forms of reading the passphrase from a file you can give '-' as file name, which results in the passphrase being read from stdin and the safety-question being skipped.

You cannot call luksFormat on a device or filesystem that is mapped or in use, e.g. mounted filesysem, used in LVM, active RAID member etc. The device or filesystem has to be un-mounted in order to call luksFormat.

To use LUKS2, specify −−*type luks2*.

<options> can be [−−hash, −−cipher, −−verify−passphrase, −−key−size, −−key−slot, −−key−file (takes precedence over optional second argument), −−keyfile−offset, −−keyfile−size, −−use−random | −−use−urandom, −−uuid, −−master−key−file, −−iter−time, −−header, −−pbkdf−force−iterations, −−force−password, −−disable-locks].

For LUKS2, additional **<options>** can be [−−integrity, −−integrity−no−wipe, −−sector−size, −−label, −−subsystem, −−pbkdf, −−pbkdf−memory, −−pbkdf−parallel, −−disable−locks, −−disable−keyring, −−luks2−metadata−size, −−luks2−keyslots−size, −−keyslot−cipher, −−keyslot−key−size].

WARNING: Doing a luksFormat on an existing LUKS container will make all data the old container permanently irretrievable unless you have a header backup.

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.

First, the passphrase is searched in LUKS tokens. If it's not found in any token and also the passphrase is not supplied via −−key-file, the command prompts for it interactively.

<options> can be [−−key−file, −−keyfile−offset, −−keyfile−size, −−readonly, −−test−passphrase, −−allow−discards, −−header, −−key-slot, −−master−key−file, −−token−id, −−token−only, −−token-type, −−disable−external−tokens, −−disable−keyring, −−disable−locks, −−type, −−refresh, −−serialize−memory−hard−pbkdf].

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. Needs kernel 2.6.19 or later.

After this operation you have to use *luksResume* to reinstate the encryption key and unblock the device or *close* to remove the mapped device.

WARNING: never suspend the device on which the cryptsetup binary resides.

<options> can be [−−header, −−disable−locks].

luksResume <name>

Resumes a suspended device and reinstates the encryption key. Prompts interactively for a passphrase if −−key-file is not given.

<options> can be [−−key−file, −−keyfile−size, −−header, −−disable−keyring, −−disable−locks, −−type]

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

Adds a new passphrase. An existing passphrase must be supplied interactively or via −−key-file. The new passphrase to be added can be specified interactively or read from the file given as positional argument.

NOTE: with −−unbound option the action creates new unbound LUKS2 keyslot. The keyslot cannot be used for device activation. If you don't pass new key via −−master−key−file option, new random key is generated. Existing passphrase for any active keyslot is not required.

<options> can be [−−key−file, −−keyfile−offset, −−keyfile−size, −−new−keyfile−offset, −−new−keyfile−size, −−key−slot, −−master−key−file, −−force−password, −−header, −−disable−locks, −−iter-time, −−pbkdf, −−pbkdf−force−iterations, −−unbound, −−type, −−keyslot−cipher, −−keyslot−key−size].

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

Removes the supplied passphrase from the LUKS device. The passphrase to be removed can be specified interactively, as the positional argument or via −−key-file.

<options> can be [−−key−file, −−keyfile−offset, −−keyfile−size, −−header, −−disable−locks, −−type]

WARNING: If you read the passphrase from stdin (without further argument or with '-' as an argument to −−key−file), batch-mode (−q) will be implicitly switched on and no warning will be given when you remove the last remaining passphrase from a LUKS container. Removing the last passphrase makes the LUKS container permanently inaccessible.

luksChangeKey <device> [<new key file>]

Changes an existing passphrase. The passphrase to be changed must be supplied interactively or via −−key−file. The new passphrase can be supplied interactively or in a file given as positional argument.

If a key-slot is specified (via −−key-slot), the passphrase for that key-slot must be given and the new passphrase will overwrite the specified key-slot. If no key-slot is specified and there is still a free key-slot, then the new passphrase will be put into a free key-slot before the key-slot containing the old passphrase is purged. If there is no free key-slot, then the key-slot with the old passphrase is overwritten directly.

WARNING: If a key-slot is overwritten, a media failure during this operation can cause the overwrite to fail after the old passphrase has been wiped and make the LUKS container inaccessible.

<options> can be [−−key−file, −−keyfile−offset, −−keyfile−size, −−new−keyfile−offset, −−iter-time, −−pbkdf, −−pbkdf−force−iterations, −−new−keyfile−size, −−key−slot, −−force−password, −−header, −−disable−locks, −−type, −−keyslot−cipher, −−keyslot−key−size].

luksConvertKey <device>

Converts an existing LUKS2 keyslot to new pbkdf parameters. The passphrase for keyslot to be converted must be supplied interactively or via −−key−file. If no −−pbkdf parameters are specified LUKS2 default pbkdf values will apply.

If a keyslot is specified (via −−key−slot), the passphrase for that keyslot must be given. If no keyslot is specified and there is still a free keyslot, then the new parameters will be put into a free keyslot before the keyslot containing the old parameters is purged. If there is no free keyslot, then the keyslot with the old parameters is overwritten directly.

WARNING: If a keyslot is overwritten, a media failure during this operation can cause the overwrite to fail after the old parameters have been wiped and make the LUKS container inaccessible.

<options> can be [−−key−file, −−keyfile−offset, −−keyfile−size, −−key−slot, −−header, −−disable−locks, −−iter-time, −−pbkdf, −−pbkdf−force−iterations, −−pbkdf−memory, −−pbkdf−parallel, −−keyslot−cipher, −−keyslot−key−size].

luksKillSlot <device> <key slot number>

Wipe the key-slot number <key slot> from the LUKS device. Except running in batch-mode (−q) a remaining passphrase must be supplied, either interactively or via −−key-file. This command can remove the last remaining key-slot, but requires an interactive confirmation when doing so. Removing the last passphrase makes a LUKS container permanently inaccessible.

<options> can be [−−key−file, −−keyfile−offset, −−keyfile−size, −−header, −−disable−locks, −−type].

WARNING: If you read the passphrase from stdin (without further argument or with '-' as an argument to −−key-file), batch-mode (−q) will be implicitly switched on and no warning will be given when you remove the last remaining passphrase from a LUKS container. Removing the last passphrase makes the LUKS container permanently inaccessible.

NOTE: If there is no passphrase provided (on stdin or through −−key-file argument) and batch-mode (−q) is active, the key-slot is removed without any other warning.

erase <device>
luksErase <device>

Erase all keyslots and make the LUKS container permanently inaccessible. You do not need to provide any password for this operation.

WARNING: This operation is irreversible.

luksUUID <device>

Print the UUID of a LUKS device.
Set new UUID if −−*uuid* option is specified.

isLuks <device>

Returns true, if <device> is a LUKS device, false otherwise. Use option −v to get human-readable feedback. 'Command successful.' means the device is a LUKS device.

By specifying −−type you may query for specific LUKS version.

luksDump <device>

Dump the header information of a LUKS device.

If the −−dump−master−key option is used, the LUKS device master key is dumped instead of the keyslot info. Together with −−master−key−file option, master key is dumped to a file instead of standard output. Beware that the master key cannot be changed without reencryption and can be used to decrypt the data stored in the LUKS container without a passphrase and even without the LUKS header. This means that if the master key is compromised, the whole device has to be erased or reencrypted to prevent further access. Use this option carefully.

To dump the master key, a passphrase has to be supplied, either interactively or via −−key−file.

To dump unbound key (LUKS2 format only), −−unbound parameter, specific −−key-slot id and proper passphrase has to be supplied, either interactively or via −−key−file. Optional −−master−key−file parameter enables unbound keyslot dump to a file.

To dump LUKS2 JSON metadata (without basic heade information like UUID) use −−dump−json−metadata option.

<options> can be [−−dump−master−key, −−dump−json−metadata, −−key−file, −−keyfile−offset,

−−keyfile−size, −−header, −−disable−locks, −−master−key−file, −−type, −−unbound, −−key-slot].

WARNING: If −−dump−master−key is used with −−key−file and the argument to −−key−file is '-', no validation question will be asked and no warning given.

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

Stores a binary backup of the LUKS header and keyslot area.
Note: Using '-' as filename writes the header backup to a file named '-'.

WARNING: This backup file and a passphrase valid at the time of backup allows decryption of the LUKS data area, even if the passphrase was later changed or removed from the LUKS device. Also note that with a header backup you lose the ability to securely wipe the LUKS device by just overwriting the header and key-slots. You either need to securely erase all header backups in addition or overwrite the encrypted data area as well. The second option is less secure, as some sectors can survive, e.g. due to defect management.

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

Restores a binary backup of the LUKS header and keyslot area from the specified file.
Note: Using '-' as filename reads the header backup from a file named '-'.

WARNING: Header and keyslots will be replaced, only the passphrases from the backup will work afterward.

This command requires that the master key size and data offset of the LUKS header already on the device and of the header backup match. Alternatively, if there is no LUKS header on the device, the backup will also be written to it.

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

Action *add* creates new keyring token to enable auto-activation of the device. For the auto-activation, the passphrase must be stored in keyring with the specified description. Usually, the passphrase should be stored in *user* or *user-session* keyring. The*tok en* command is supported only for LUKS2.

For adding new keyring token, option −−key−description is mandatory. Also, new token is assigned to key slot specified with −−key−slot option or to all active key slots in the case −−key−slot option is omitted.

To remove existing token, specify the token ID which should be removed with −−token−id option.

WARNING: The action *token remove* removes any token type, not just *keyring* type from token slot specified by −−token−id option.

Action *import* can store arbitrary valid token json in LUKS2 header. It may be passed via standard input or via file passed in −−json−file option. If you specify −−key−slot then successfully imported token is also assigned to the key slot.

Action *export* writes requested token json to a file passed with −−json−file or to standard output.

<options> can be [−−header, −−token−id, −−key−slot, −−key−description, −−disable−external−tokens, −−disable−locks, −−disable−keyring, −−json−file].

convert <device> −−type <format>

Converts the device between LUKS1 and LUKS2 format (if possible). The conversion will not be performed if there is an additional LUKS2 feature or LUKS1 has unsupported header size.

Conversion (both directions) must be performed on inactive device. There must not be active dm-crypt mapping established for LUKS header requested for conversion.

−−**type** option is mandatory with following accepted values: *luks1* or *luks2*.

WARNING: The *convert* action can destroy the LUKS header in the case of a crash during conversion or if a media error occurs. Always create a header backup before performing this operation!

<options> can be [−−header, −−type].

config <device>

Set permanent configuration options (store to LUKS header). The *config* command is supported only for LUKS2.

The permanent options can be −−*priority* to set priority (normal, prefer, ignore) for keyslot (specified by −−*key−slot*) or −−*label* and −−*subsystem*.

<options> can be [−−priority, −−label, −−subsystem, −−key−slot, −−header].

loop-AES EXTENSION

cryptsetup supports mapping loop-AES encrypted partition using a compatibility mode.

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>.

If the key file is encrypted with GnuPG, then you have to use −−key−file=− and decrypt it before use, e.g. like this:
gpg −−decrypt <keyfile> | cryptsetup loopaesOpen −−key−file=− <device> <name>

WARNING: The loop-AES extension cannot use the direct input of key file on real terminal because the keys are separated by end-of-line and only part of the multi-key file would be read.
If you need it in script, just use the pipe redirection:
echo $keyfile | cryptsetup loopaesOpen −−key−file=− <device> <name>

Use −−**keyfile−size** to specify the proper key length if needed.

Use −−**offset** to specify device offset. Note that the units need to be specified in number of 512 byte sectors.

Use −−**skip** to specify the IV offset. If the original device used an offset and but did not use it in IV sector calculations, you have to explicitly use −−**skip 0** in addition to the offset parameter.

Use −−**hash** to override the default hash function for passphrase hashing (otherwise it is detected according to key size).

<options> can be [−−key−file, −−key−size, −−offset, −−skip, −−hash, −−readonly, −−allow−discards, −−refresh].

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

TCRYPT (TrueCrypt-compatible and VeraCrypt) 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 just extension of TrueCrypt header with increased iteration count so unlocking can take quite a lot of time (in comparison with TCRYPT device).

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

Opens the TCRYPT (a TrueCrypt-compatible) <device> and sets up a mapping <name>.

<options> can be [−−key−file, −−tcrypt−hidden, −−tcrypt−system, −−tcrypt−backup, −−readonly, −−test−passphrase, −−allow-discards, −−disable−veracrypt, −−veracrypt−pim, −−veracrypt−query−pim, −−header, −−cipher, −−hash].

The keyfile parameter allows a combination of file content with the passphrase and can be repeated. Note that using keyfiles is compatible with TCRYPT and is different from LUKS keyfile logic.

If **−−PBKDF2 variants with the specified hash algorithms are checked. This could speed up unlocking the device (but also it reveals some information about the container).**

If you use −−header in combination with hidden or system options, the header file must contain specific headers on the same positions as the original encrypted container.

WARNING: Option **−−allow−discards** cannot be combined with option **−−tcrypt−hidden**. For normal mapping, it can cause the **destruction of hidden volume** (hidden volume appears as unused space for outer volume so this space can be discarded).

tcryptDump <device>

Dump the header information of a TCRYPT device.

If the −−dump−master−key option is used, the TCRYPT device master key is dumped instead of TCRYPT header info. Beware that the master key (or concatenated master keys if cipher chain is used) can be used to decrypt the data stored in the TCRYPT container without a passphrase. This means that if the master key is compromised, the whole device has to be erased to prevent further access. Use this option carefully.

<options> can be [−−dump−master−key, −−key−file, −−tcrypt−hidden, −−tcrypt−system, −−tcrypt−backup, −−cipher, −−hash].

The keyfile parameter allows a combination of file content with the passphrase and can be repeated.

See also **https://en.wikipedia.org/wiki/TrueCrypt** for more information regarding TrueCrypt.

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

BITLK (Windows BitLocker-compatible) EXTENSION (EXPERIMENTAL)

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.

WARNING: This extension is EXPERIMENTAL.

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 master key is supported. You must provide BitLocker Full Volume Encryption Key (FVEK) using the −−master−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 −−type bitlk <device> <name>
bitlkOpen <device> <name> (**old syntax**)

Opens the BITLK (a BitLocker-compatible) <device> and sets up a mapping <name>.

<options> can be [−−key−file, −−readonly, −−test−passphrase, −−allow−discards −−master−key−file].

bitlkDump <device>

Dump the header information of a BITLK device.

<options> can be [−−dump−master−key −−master−key−file].

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

MISCELLANEOUS

repair <device>

Tries to repair the device metadata if possible. Currently supported only for LUKS device type.

This command is useful to fix some known benign LUKS metadata header corruptions. Only basic corruptions of unused keyslot are fixable. This command will only change the LUKS header, not any key-slot data. You may enforce LUKS version by adding −−type option.

It also repairs (upgrades) LUKS2 reencryption metadata by adding metadata digest that protects it against malicious changes.

If LUKS2 reencryption was interrupted in the middle of writting reencryption segment the repair command can be used to perform reencryption recovery so that reencryption can continue later.

WARNING: Always create a binary backup of the original header before calling this command.

benchmark <options>

Benchmarks ciphers and KDF (key derivation function). Without parameters, it tries to measure few common configurations.

To benchmark other ciphers or modes, you need to specify **−−cipher** and **−−key−size** options or **−−hash** for KDF test.

NOTE: This benchmark is using memory only and is only informative. You cannot directly predict real storage encryption speed from it.

For testing block ciphers, this benchmark 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).

<options> can be [−−cipher, −−key−size, −−hash].

OPTIONS

−−verbose, −v
Print more information on command execution.

−−debug or −−debug−json
Run in debug mode with full diagnostic logs. Debug output lines are always prefixed by '#'. If −−debug−json is used, additional LUKS2 JSON data structures are printed.

−−type <device-type>
Specifies required device type, for more info read *BASIC ACTIONS* section.

−−hash, −h <*hash−spec*>
Specifies the passphrase hash for *open* (for plain and loopaes device types).

Specifies the hash used in the LUKS key setup scheme and volume key digest for *luksFormat*. The specified hash is used as hash-parameter for PBKDF2 and for the AF splitter.

The specified hash name is passed to the compiled-in crypto backend. Different backends may support different hashes. For *luksFormat*, the hash algorithm must provide at least 160 bits of output, which excludes, e.g., MD5. Do not use a non-crypto hash like **"crc32"** as this breaks security.

Values compatible with old version of cryptsetup are **"ripemd160"** for *open −−type plain* and **"sha1"** for *luksFormat*.

Use *cryptsetup −−help* to show the defaults.

−−cipher, −c <*cipher−spec*>
Set the cipher specification string.

cryptsetup −−help shows the compiled-in defaults. The current default in the distributed sources is "aes-cbc-essiv:sha256" for plain dm-crypt and "aes-xts-plain64" for LUKS.

If a hash is part of the cipher specification, then it is used as part of the IV generation. For example, ESSIV needs a hash function, while "plain64" does not and hence none is specified.

For XTS mode you can optionally set a key size of 512 bits with the −s option. Key size for XTS mode is twice that for other modes for the same security level.

XTS mode requires kernel 2.6.24 or later and plain64 requires kernel 2.6.33 or later. More information can be found in the FAQ.

−−verify-passphrase, −y
When interactively asking for a passphrase, ask for it twice and complain if both inputs do not match. Advised when creating a regular mapping for the first time, or when running *luksFormat*. Ignored on input from file or stdin.

−−key-file, −d *name*
Read the passphrase from file.

If the name given is "-", then the passphrase will be read from stdin. In this case, reading will not stop at newline characters.

With LUKS, passphrases supplied via −−key−file are always the existing passphrases requested by

a command, except in the case of *luksFormat* where −−key−file is equivalent to the positional key file argument.

If you want to set a new passphrase via key file, you have to use a positional argument to *luksAddKey*.

See section **NOTES ON PASSPHRASE PROCESSING** for more information.

−−keyfile−offset *value*
Skip *value* bytes at the beginning of the key file. Works with all commands that accept key files.

−−keyfile−size, −l *value*
Read a maximum of *value* bytes from the key file. The default is to read the whole file up to the compiled-in maximum that can be queried with −−help. Supplying more data than the compiled-in maximum aborts the operation.

This option is useful to cut trailing newlines, for example. If −−keyfile−offset is also given, the size count starts after the offset. Works with all commands that accept key files.

−−new−keyfile−offset *value*
Skip *value* bytes at the start when adding a new passphrase from key file with *luksAddKey*.

−−new−keyfile−size *value*
Read a maximum of *value* bytes when adding a new passphrase from key file with *luksAddKey*. The default is to read the whole file up to the compiled-in maximum length that can be queried with −−help. Supplying more than the compiled in maximum aborts the operation. When −−new−keyfile−offset is also given, reading starts after the offset.

−−master−key−file
Use a master key stored in a file.

For *luksFormat* this allows creating a LUKS header with this specific master key. If the master key was taken from an existing LUKS header and all other parameters are the same, then the new header decrypts the data encrypted with the header the master key was taken from.

Action *luksDump* together with −−dump−master−key option: The volume (master) key is stored in a file instead of being printed out to standard output.

WARNING: If you create your own master key, you need to make sure to do it right. Otherwise, you can end up with a low-entropy or otherwise partially predictable master key which will compromise security.

For *luksAddKey* this allows adding a new passphrase without having to know an existing one.

For *open* this allows one to open the LUKS device without giving a passphrase.

−−dump−json−metadata
For *luksDump* (LUKS2 only) this option prints content of LUKS2 header JSON metadata area.

−−dump−master−key
For *luksDump* this option includes the master key in the displayed information. Use with care, as the master key can be used to bypass the passphrases, see also option −−master−key−file.

−−json−file
Read token json from a file or write token to it. See *token* action for more information. −−json−file=- reads json from standard input or writes it to standard output respectively.

−−use−random

−−use−urandom
For *luksFormat* these options define which kernel random number generator will be used to create the master key (which is a long-term key).

See **NOTES ON RANDOM NUMBER GENERATORS** for more information. Use *cryptsetup −−help* to show the compiled-in default random number generator.

WARNING: In a low-entropy situation (e.g. in an embedded system), both selections are problematic. Using /dev/urandom can lead to weak keys. Using /dev/random can block a long time, potentially forever, if not enough entropy can be harvested by the kernel.

−−key−slot, −S <0−N>
For LUKS operations that add key material, this options allows you to specify which key slot is selected for the new key. This option can be used for*luksF ormat*, and *luksAddKey*.
In addition, for *open*, this option selects a specific key-slot to compare the passphrase against. If the given passphrase would only match a different key-slot, the operation fails.

Maximum number of key slots depends on LUKS version. LUKS1 can have up to 8 key slots. LUKS2 can have up to 32 key slots based on key slot area size and key size, but a valid key slot ID can always be between 0 and 31 for LUKS2.

−−key−size, −s <bits>
Sets key size in bits. The argument has to be a multiple of 8. The possible key-sizes are limited by the cipher and mode used.

See /proc/crypto for more information. Note that key-size in /proc/crypto is stated in bytes.

This option can be used for *open −−type plain* or *luksFormat*. All other LUKS actions will use the key-size specified in the LUKS header. Use*cryptsetup −−help* to sho w the compiled-in defaults.

−−size, −b <number of 512 byte sectors>
Set the size of the device in sectors of 512 bytes. This option is only relevant for the *open* and *resize* actions.

−−offset, −o <number of 512 byte sectors>
Start offset in the backend device in 512-byte sectors. This option is only relevant for the *open* action with plain or loopaes device types or for LUKS devices in *luksFormat*.

For LUKS, the −−offset option sets the data offset (payload) of data device and must be be aligned to 4096-byte sectors (must be multiple of 8). This option cannot be combined with −−align−payload option.

−−skip, −p <number of 512 byte sectors>
Start offset used in IV calculation in 512-byte sectors (how many sectors of the encrypted data to skip at the beginning). This option is only relevant for the *open* action with plain or loopaes device types.

Hence, if −−offset *n*, and −−skip *s*, sector *n* (the first sector of the encrypted device) will get a sector number of *s* for the IV calculation.

−−device−size *size[units]*
Instead of real device size, use specified value.

With *reencrypt* action it means that only specified area (from the start of the device to the specified size) will be reencrypted.

With *resize* action it sets new size of the device.

If no unit suffix is specified, the size is in bytes.

Unit suffix can be S for 512 byte sectors, K/M/G/T (or KiB,MiB,GiB,TiB) for units with 1024 base or KB/MB/GB/TB for 1000 base (SI scale).

WARNING: This is destructive operation when used with reencrypt command.

−−readonly, −r
set up a read-only mapping.

−−shared
Creates an additional mapping for one common ciphertext device. Arbitrary mappings are supported. This option is only relevant for the *open −−type plain* action. Use −−offset, −−size and −−skip to specify the mapped area.

−−pbkdf <PBKDF spec>
Set Password-Based Key Derivation Function (PBKDF) algorithm for LUKS keyslot. The PBKDF can be: *pbkdf2* (for PBKDF2 according to RFC2898), *argon2i* for Argon2i or *argon2id* for Argon2id (see https://www.cryptolux.org/index.php/Argon2 for more info).

For LUKS1, only PBKDF2 is accepted (no need to use this option). The default PBKDF2 for LUKS2 is set during compilation time and is available in *cryptsetup −−help* output.

A PBKDF is used for increasing dictionary and brute-force attack cost for keyslot passwords. The parameters can be time, memory and parallel cost.

For PBKDF2, only time cost (number of iterations) applies. For Argon2i/id, there is also memory cost (memory required during the process of key derivation) and parallel cost (number of threads that run in parallel during the key derivation.

Note that increasing memory cost also increases time, so the final parameter values are measured by a benchmark. The benchmark tries to find iteration time (*−−iter−time*) with required memory cost *−−pbkdf−memory*. If it is not possible, the memory cost is decreased as well. The parallel cost *−−pbkdf−parallel* is constant and is checked against available CPU cores.

You can see all PBKDF parameters for particular LUKS2 keyslot with *luksDump* command.

NOTE: If you do not want to use benchmark and want to specify all parameters directly, use *−−pbkdf−force−iterations* with *−−pbkdf−memory* and *−−pbkdf−parallel*. This will override the values without benchmarking. Note it can cause extremely long unlocking time. Use only in specific cases, for example, if you know that the formatted device will be used on some small embedded system.

MINIMAL AND MAXIMAL PBKDF COSTS: For **PBKDF2**, the minimum iteration count is 1000 and maximum is 4294967295 (maximum for 32bit unsigned integer). Memory and parallel costs are unused for PBKDF2. For **Argon2i** and **Argon2id**, minimum iteration count (CPU cost) is 4 and maximum is 4294967295 (maximum for 32bit unsigned integer). Minimum memory cost is 32 KiB and maximum is 4 GiB. (Limited by addresable memory on some CPU platforms.) If the memory cost parameter is benchmarked (not specified by a parameter) it is always in range from 64 MiB to 1 GiB. The parallel cost minimum is 1 and maximum 4 (if enough CPUs cores are available, otherwise it is decreased).

−−iter−time, −i <number of milliseconds>
The number of milliseconds to spend with PBKDF passphrase processing. This option is only relevant for LUKS operations that set or change passphrases, such as *luksFormat* or *luksAddKey*. Specifying 0 as parameter selects the compiled-in default.

−−pbkdf−memory <number>
Set the memory cost for PBKDF (for Argon2i/id the number represents kilobytes). Note that it is maximal value, PBKDF benchmark or available physical memory can decrease it. This option is not available for PBKDF2.

−−pbkdf−parallel <number>
Set the parallel cost for PBKDF (number of threads, up to 4). Note that it is maximal value, it is decreased automatically if CPU online count is lower. This option is not available for PBKDF2.

−−pbkdf−force−iterations <num>
Avoid PBKDF benchmark and set time cost (iterations) directly. It can be used for LUKS/LUKS2 device only. See *−−pbkdf* option for more info.

−−batch−mode, −q
Suppresses all confirmation questions. Use with care!

If the −y option is not specified, this option also switches off the passphrase verification for *luksFormat*.

−−progress-frequency <seconds>
Print separate line every <seconds> with wipe progress.

−−timeout, −t <number of seconds>
The number of seconds to wait before timeout on passphrase input via terminal. It is relevant every time a passphrase is asked, for example for *open*, *luksFormat* or *luksAddKey*. It has no effect if used in conjunction with −−key-file.
This option is useful when the system should not stall if the user does not input a passphrase, e.g. during boot. The default is a value of 0 seconds, which means to wait forever.

−−tries, −T
How often the input of the passphrase shall be retried. This option is relevant every time a passphrase is asked, for example for *open*, *luksFormat* or *luksAddKey*. The default is 3 tries.

−−align−payload <number of 512 byte sectors>
Align payload at a boundary of *value* 512-byte sectors. This option is relevant for *luksFormat*.

If not specified, cryptsetup tries to use the topology info provided by the kernel for the underlying device to get the optimal alignment. If not available (or the calculated value is a multiple of the default) data is by default aligned to a 1MiB boundary (i.e. 2048 512-byte sectors).

For a detached LUKS header, this option specifies the offset on the data device. See also the −−header option.

WARNING: This option is DEPRECATED and has often unexpected impact to the data offset and keyslot area size (for LUKS2) due to the complex rounding. For fixed data device offset use *−−offset* option instead.

−−uuid=*UUID*
Use the provided *UUID* for the *luksFormat* command instead of generating a new one. Changes the existing UUID when used with the *luksUUID* command.

The UUID must be provided in the standard UUID format, e.g. 12345678-1234-1234-1234-123456789abc.

−−allow−discards
Allow the use of discard (TRIM) requests for the device. This option is only relevant for *open* action. This is also not supported for LUKS2 devices with data integrity protection.

WARNING: This command can have a negative security impact because it can make filesystem-

level operations visible on the physical device. For example, information leaking filesystem type, used space, etc. may be extractable from the physical device if the discarded blocks can be located later. If in doubt, do not use it.

A kernel version of 3.1 or later is needed. For earlier kernels, this option is ignored.

−−perf−same_cpu_crypt
Perform encryption using the same cpu that IO was submitted on. The default is to use an unbound workqueue so that encryption work is automatically balanced between available CPUs. This option is only relevant for *open* action.

NOTE: This option is available only for low-level dm-crypt performance tuning, use only if you need a change to default dm-crypt behaviour. Needs kernel 4.0 or later.

−−perf−submit_from_crypt_cpus
Disable offloading writes to a separate thread after encryption. There are some situations where offloading write bios from the encryption threads to a single thread degrades performance significantly. The default is to offload write bios to the same thread. This option is only relevant for *open* action.

NOTE: This option is available only for low-level dm-crypt performance tuning, use only if you need a change to default dm-crypt behaviour. Needs kernel 4.0 or later.

−−perf−no_read_workqueue, −−perf−no_write_workqueue
Bypass dm-crypt internal workqueue and process read or write requests synchronously. This option is only relevant for *open* action.

NOTE: These options are available only for low-level dm-crypt performance tuning, use only if you need a change to default dm-crypt behaviour. Needs kernel 5.9 or later.

−−test−passphrase
Do not activate the device, just verify passphrase. This option is only relevant for *open* action (the device mapping name is not mandatory if this option is used).

−−header <device or file storing the LUKS header>
Use a detached (separated) metadata device or file where the LUKS header is stored. This option allows one to store ciphertext and LUKS header on different devices.

This option is only relevant for LUKS devices and can be used with the *luksFormat*, *open*, *luksSuspend*, *luksResume*, *status* and *resize* commands.

For *luksFormat* with a file name as the argument to −−header, the file will be automatically created if it does not exist. See the cryptsetup FAQ for header size calculation.

For other commands that change the LUKS header (e.g. *luksAddKey*), specify the device or file with the LUKS header directly as the LUKS device.

If used with *luksFormat*, the −−align−payload option is taken as absolute sector alignment on ciphertext device and can be zero.

WARNING: There is no check whether the ciphertext device specified actually belongs to the header given. In fact, you can specify an arbitrary device as the ciphertext device for *open* with the −−header option. Use with care.

−−header−backup−file <file>
Specify file with header backup for *luksHeaderBackup* or *luksHeaderRestore* actions.

−−force−password
Do not use password quality checking for new LUKS passwords.

This option applies only to *luksFormat*, *luksAddKey* and *luksChangeKey* and is ignored if cryptsetup is built without password quality checking support.

For more info about password quality check, see the manual page for **pwquality.conf(5)** and **passwdqc.conf(5)**.

−−deferred
Defers device removal in *close* command until the last user closes it.

−−cancel−deferred
Removes a previously configured deferred device removal in *close* command.

−−disable−external−tokens
Disable loading of plugins for external LUKS2 tokens.

−−disable−locks
Disable lock protection for metadata on disk. This option is valid only for LUKS2 and ignored for other formats.

WARNING: Do not use this option unless you run cryptsetup in a restricted environment where locking is impossible to perform (where /run directory cannot be used).

−−disable−keyring
Do not load volume key in kernel keyring and store it directly in the dm-crypt target instead. This option is supported only for the LUKS2 format.

−−key−description <text>
Set key description in keyring for use with *token* command.

−−priority <normal|prefer|ignore>
Set a priority for LUKS2 keyslot. The*pr efer* priority marked slots are tried before *normal* priority. The *ignored* priority means, that slot is never used, if not explicitly requested by *−−key−slot* option.

−−token−id
Specify what token to use in actions *token*, *open* or *resize*. If omitted, all available tokens will be checked before proceeding further with passphrase prompt.

−−token−only
Do not proceed further with action (any of *token*, *open* or *resize*) if token activation failed. Without the option, action asks for passphrase to proceed further.

−−token−type
Restrict tokens eligible for operation to specific token type (name). Mostly useful when no −−token−id is specified.

−−sector−size <bytes>
Set sector size for use with disk encryption. It must be power of two and in range 512 - 4096 bytes. This option is available only in the LUKS2 or plain modes.

The default for plain mode is 512 bytes. For LUKS2 devices it's established during luksFormat operation based on parameters provided by underlying data device. For native 4K block devices it's 4096 bytes. For 4K/512e (4K physical sector size with 512 bytes emulation) it's 4096 bytes. For drives reporting only 512 bytes block size it remains 512 bytes. If data device is regular file put in filesystem it's 4096 bytes.

Note that if sector size is higher than underlying device hardware sector and there is not integrity protection that uses data journal, using this option can increase risk on incomplete sector writes

during a power fail.

If used together with *−−integrity* option and dm-integrity journal, the atomicity of writes is guaranteed in all cases (but it cost write performance - data has to be written twice).

Increasing sector size from 512 bytes to 4096 bytes can provide better performance on most of the modern storage devices and also with some hw encryption accelerators.

−−iv-large-sectors
Count Initialization Vector (IV) in larger sector size (if set) instead of 512 bytes sectors. This option can be used only for *open* command and *plain* encryption type.

NOTE: This option does not have any performance or security impact, use it only for accessing incompatible existing disk images from other systems that require this option.

−−persistent
If used with LUKS2 devices and activation commands like *open* or *refresh*, the specified activation flags are persistently written into metadata and used next time automatically even for normal activation. (No need to use cryptab or other system configuration files.)

If you need to remove a persistent flag, use *−−persistent* without the flag you want to remove (e.g. to disable persistently stored discard flag, use *−−persistent* without *−−allow-discards*).

Only *−−allow-discards*, *−−perf−same_cpu_crypt*, *−−perf−submit_from_crypt_cpus*, *−−perf−no_read_workqueue*, *−−perf−no_write_workqueue* and *−−integrity−no−journal* can be stored persistently.

−−refresh
Refreshes an active device with new set of parameters. See action *refresh* description for more details.

−−label <LABEL>
−−subsystem <SUBSYSTEM> Set label and subsystem description for LUKS2 device, can be used in *config* and *format* actions. The label and subsystem are optional fields and can be later used in udev scripts for triggering user actions once device marked by these labels is detected.

−−integrity <integrity algorithm>
Specify integrity algorithm to be used for authenticated disk encryption in LUKS2.

WARNING: This extension is EXPERIMENTAL and requires dm-integrity kernel target (available since kernel version 4.12). For native AEAD modes, also enable "User-space interface for AEAD cipher algorithms" in "Cryptographic API" section (CONFIG_CRYPTO_USER_API_AEAD .config option).

For more info, see *AUTHENTICATED DISK ENCRYPTION* section.

−−luks2−metadata−size <size>
This option can be used to enlarge the LUKS2 metadata (JSON) area. The size includes 4096 bytes for binary metadata (usable JSON area is smaller of the binary area). According to LUKS2 specification, only these values are valid: 16, 32, 64, 128, 256, 512, 1024, 2048 and 4096 kB The <size> can be specified with unit suffix (for example 128k).

−−luks2−keyslots−size <size>
This option can be used to set specific size of the LUKS2 binary keyslot area (key material is encrypted there). The value must be aligned to multiple of 4096 bytes with maximum size 128MB. The <size> can be specified with unit suffix (for example 128k).

−−keyslot−cipher <cipher−spec>
This option can be used to set specific cipher encryption for the LUKS2 keyslot area.

−−keyslot−key−size <bits>
This option can be used to set specific key size for the LUKS2 keyslot area.

−−integrity−no−journal
Activate device with integrity protection without using data journal (direct write of data and integrity tags). Note that without journal power fail can cause non-atomic write and data corruption. Use only if journalling is performed on a different storage layer.

−−integrity−no−wipe
Skip wiping of device authentication (integrity) tags. If you skip this step, sectors will report invalid integrity tag until an application write to the sector.

NOTE: Even some writes to the device can fail if the write is not aligned to page size and page-cache initiates read of a sector with invalid integrity tag.

−−unbound

Creates new or dumps existing LUKS2 unbound keyslot. See *luksAddKey* or *luksDump* actions for more details.

−−tcrypt−hidden
−−tcrypt−system −−tcrypt−backup Specify which TrueCrypt on-disk header will be used to open the device. See*TCR YPT* section for more info.

−−veracrypt
This option is ignored as VeraCrypt compatible mode is supported by default.

−−disable−veracrypt
This option can be used to disable VeraCrypt compatible mode (only TrueCrypt devices are recognized). Only for TCRYPT extension. See *TCRYPT* section for more info.

−−veracrypt−pim
−−veracrypt−query−pim Use a custom Personal Iteration Multiplier (PIM) for VeraCrypt device. See *TCRYPT* section for more info.

−−serialize−memory−hard−pbkdf
Use a global lock to serialize unlocking of keyslots using memory-hard PBKDF.

NOTE: This is (ugly) workaround for a specific situation when multiple devices are activated in parallel and system instead of reporting out of memory starts unconditionally stop processes using out-of-memory killer.

DO NOT USE this switch until you are implementing boot environment with parallel devices activation!

−−encrypt
Initialize (and run) device encryption (*reencrypt* action parameter)

−−decrypt
Initialize (and run) device decryption (*reencrypt* action parameter)

−−init−only
Initialize reencryption (any variant) operation in LUKS2 metadata only and exit. If any reencrypt operation is already initialized in metadata, the command with −−init−only parameter fails.

−−resume−only
Resume reencryption (any variant) operation already described in LUKS2 metadata. If no reencrypt operation is initialized, the command with −−resume−only parameter fails. Useful for

resuming reencrypt operation without accidentally triggering new reencryption operation.

−−resilience <mode>
Reencryption resilience mode can be one of *checksum*, *journal* or *none*.

checksum: default mode, where individual checksums of ciphertext hotzone sectors are stored, so the recovery process can detect which sectors were already reencrypted. It requires that the device sector write is atomic.

journal: the hotzone is journaled in the binary area (so the data are written twice).

none: performance mode. There is no protection and the only way it's safe to interrupt the reencryption is similar to old offline reencryption utility. (ctrl+c).

The option is ignored if reencryption with datashift mode is in progress.

−−resilience-hash <hash>
The hash algorithm used with "−−resilience checksum" only. The default hash is sha256. With other resilience modes, the hash parameter is ignored.

−−hotzone-size <size>
This option can be used to set an upper limit on the size of reencryption area (hotzone). The <size> can be specified with unit suffix (for example 50M). Note that actual hotzone size may be less than specified <size> due to other limitations (free space in keyslots area or available memory).

−−reduce−device−size <size>
Initialize LUKS2 reencryption with data device size reduction (currently only −−encrypt variant is supported).

Last <size> sectors of <device> will be used to properly initialize device reencryption. That means any data at last <size> sectors will be lost.

It could be useful if you added some space to underlying partition or logical volume (so last <size> sectors contains no data).

Recommended minimal size is twice the default LUKS2 header size (−−reduce−device−size 32M) for −−encrypt use case. Be sure to have enough (at least −−reduce−device−size value of free space at the end of <device>).

WARNING: This is a destructive operation and cannot be reverted. Use with extreme care - accidentally overwritten filesystems are usually unrecoverable.

−−version
Show the program version.

−−usage
Show short option help.

−−help, −?
Show help text and default parameters.

EXAMPLE

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 afterwards 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 ON 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.

NOTES ON PASSPHRASE PROCESSING FOR LUKS

LUKS uses PBKDF2 to protect against dictionary attacks and to give some protection to low-entropy passphrases (see RFC 2898 and the 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.

NOTES ON 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 (master) key. You can switch between using /dev/random and /dev/urandom here, see**−−use−random** and **−−use−urandom** options. Using /de v/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/lock/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.

DEPRECATED ACTIONS

The *reload* action is no longer supported. Please use *dmsetup(8)* if you need to directly manipulate with the device mapping table.

The *luksDelKey* was replaced with *luksKillSlot*.

REPORTING BUGS

Report bugs, including ones in the documentation, on the cryptsetup mailing list at <dm-crypt@saout.de> or in the 'Issues' section on LUKS website. Please attach the output of the failed command with the −−debug option added.

AUTHORS

cryptsetup 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>.

COPYRIGHT

SEE ALSO

The LUKS website at **https://gitlab.com/cryptsetup/cryptsetup/**

The cryptsetup FAQ, contained in the distribution package and online at **https://gitlab.com/cryptsetup/cryptsetup/wikis/FrequentlyAskedQuestions**

The cryptsetup mailing list and list archive, see FAQ entry 1.6.

The LUKS version 1 on-disk format specification available at **https://gitlab.com/cryptsetup/cryptsetup/wikis/Specification** and LUKS version 2 at **https://gitlab.com/cryptsetup/LUKS2-docs**.