

NAME

cryptsetup-luksAddKey – add a new passphrase

SYNOPSIS

cryptsetup luksAddKey [**<options>**] **<device>** [**<key file with new key>**]

DESCRIPTION

Adds a keyslot protected by a new passphrase. An existing passphrase must be supplied interactively, via `--key-file` or LUKS2 token (plugin). Alternatively to the existing passphrase, the user may pass directly the volume key (via `--volume-key-file` or `--volume-key-keyring`). The new passphrase to be added can be specified interactively, read from the file given as the positional argument (also via `--new-keyfile` parameter) or via LUKS2 token.

The `--unbound` option creates a new unbound LUKS2 keyslot. An unbound keyslot stores an independent key that cannot be used for device activation. A new random key is generated if you don't pass a new key via the `--volume-key-file` option. The existing passphrase for any active keyslot is not required.

Some parameters are effective only if used with the LUKS2 format that supports per-keyslot parameters. For LUKS1, the PBKDF type and hash algorithm are always the same for all keyslots.

<options> can be [`--key-file`, `--keyfile-offset`, `--keyfile-size`, `--new-keyfile`, `--new-keyfile-offset`, `--new-keyfile-size`, `--key-slot`, `--new-key-slot`, `--volume-key-file`, `--volume-key-keyring`, `--force-password`, `--hash`, `--header`, `--disable-locks`, `--iter-time`, `--pbkdf`, `--pbkdf-force-iterations`, `--pbkdf-memory`, `--pbkdf-parallel`, `--unbound`, `--type`, `--keyslot-cipher`, `--keyslot-key-size`, `--key-size`, `--timeout`, `--token-id`, `--token-type`, `--token-only`, `--new-token-id`, `--verify-passphrase`, `--external-tokens-path`].

OPTIONS**--batch-mode, -q**

Suppresses all confirmation questions. Use with care!

If the `--verify-passphrase` option is not specified, this option also switches off the passphrase verification.

--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.

--disable-locks

Disable lock protection for metadata on disk. This option is valid only for LUKS2 and is 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).

--external-tokens-path *<absolute path>*

Override the system directory path where cryptsetup searches for external token handlers (or token plugins). It must be an absolute path (starting with `/` character).

--force-password

Do not use password quality checking for new LUKS passwords.

This option 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).

--hash, -h *<hash-spec>*

The specified hash is used for PBKDF2 and the AF splitter.

--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 the ciphertext and LUKS header on different devices.

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

--help, -?

Show help text and default parameters.

--iter-time, -i *<number of milliseconds>*

The number of milliseconds to spend with PBKDF passphrase processing. Specifying 0 as a parameter selects the compiled-in default.

--key-description *text*

Set the key description in the keyring that will be used for passphrase retrieval.

--key-file, -d *file*

Read the passphrase from the file.

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

The passphrase supplied via **--key-file** is always the passphrase for the existing keyslot requested by the command.

If you want to set a new passphrase via key file, you have to use a positional argument or parameter **--new-keyfile**.

See section *NOTES ON PASSPHRASE PROCESSING* in **cryptsetup(8)** for more information.

--keyfile-offset *value*

Skip *value* bytes at the beginning of the key file.

--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.

--key-size, -s *bits*

Provide volume key size in *bits*. The argument has to be a multiple of 8.

This option is required when the parameter **--volume-key-file** is used to provide current volume key. Also, it is used when a new unbound keyslot is created by specifying **--unbound** parameter.

--key-slot, -S *<0-N>*

When used together with the parameter `--new-key-slot`, this option allows you to specify which keyslot is selected for unlocking the volume key.

This option is ignored if the existing volume key gets unlocked via LUKS2 token (`--token-id`, `--token-type` or `--token-only` parameters) or when volume key is provided directly via `--volume-key-file` parameter.

To maintain backward compatibility, without `--new-key-slot` parameter, this option allows you to specify which keyslot is selected for the new key.

The maximum number of keyslots depends on the LUKS version. LUKS1 can have up to 8 keyslots. LUKS2 can have up to 32 keyslots based on keyslot area size and key size, but a valid keyslot ID can always be between 0 and 31 for LUKS2.

--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 a specific key size for the LUKS2 keyslot area.

--new-keyfile *name*

Read the passphrase for a new keyslot from a file.

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

This is an alternative method to positional argument when adding a new passphrase via keyfile.

--new-keyfile-offset *value*

Skip *value* bytes at the start when adding a new passphrase from the key file.

--new-keyfile-size *value*

Read a maximum of *value* bytes when adding a new passphrase from the key file. 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.

--new-key-description *text*

Set the key description in the keyring that will be used for new passphrase retrieval.

--new-key-slot *<0-N>*

This option allows you to specify which keyslot is selected for the new key.

The maximum number of keyslots depends on the LUKS version. LUKS1 can have up to 8 keyslots. LUKS2 can have up to 32 keyslots based on keyslot area size and key size, but a valid keyslot ID can always be between 0 and 31 for LUKS2.

--new-token-id *<id>*

Specify what token to use to get the passphrase for a new keyslot.

--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 Argon2 <https://www.cryptolux.org/index.php/Argon2> for more info).

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

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

For PBKDF2, only the 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 a particular LUKS2 keyslot with the **cryptsetup-luksDump(8)** command.

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 or cause out-of-memory conditions with unconditional process termination. 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 the maximum is 4294967295 (maximum for 32-bit unsigned integer). Memory and parallel costs are not supported for PBKDF2. For **Argon2i** and **Argon2id**, the minimum iteration count (CPU cost) is 4, and the maximum is 4294967295 (maximum for a 32-bit unsigned integer). Minimum memory cost is 32 KiB and maximum is 4 GiB. If the memory cost parameter is benchmarked (not specified by a parameter), it is always in the range from 64 MiB to 1 GiB. Memory cost above 1GiB (up to the 4GiB maximum) can be setup only by the *--pbkdf-memory* parameter. The parallel cost minimum is 1 and maximum 4 (if enough CPU cores are available, otherwise it is decreased by the available CPU cores).

WARNING: Increasing PBKDF computational costs above the mentioned limits provides negligible additional security improvement. While elevated costs significantly increase brute-force overhead, they offer negligible protection against dictionary attacks. The marginal cost increase for processing an entire dictionary remains fundamentally insufficient.

The hardcoded PBKDF limits represent engineered trade-offs between cryptographic security and operational usability. LUKS maintains portability and must be used within a reasonable time on resource-constrained systems.

Cryptsetup deliberately restricts maximum memory cost (4 GiB) and parallel cost (4) parameters due to architectural limitations (like embedded and legacy systems).

PBKDF memory cost mandates actual physical RAM allocation with intensive write operations that must remain in physical RAM. Any swap usage results in unacceptable performance degradation. Memory management often overcommits allocations beyond available physical memory, expecting most allocated memory to remain unused. In such situations, as PBKDF always uses all allocated memory, it frequently causes out-of-memory failures that abort cryptsetup operations.

--pbkdf-force-iterations number

Avoid the PBKDF benchmark and set the time cost (iterations) directly. It can be used only for a

LUKS/LUKS2 device. See `--pbkdf` option for more info.

`--pbkdf-memory` *number*

Set the memory cost for PBKDF (for Argon2i/id, the number represents kilobytes). Note that it is the 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 the maximal value; it is decreased automatically if the CPU online count is lower. This option is not available for PBKDF2.

`--timeout, -t` *seconds*

The number of seconds to wait before a timeout on passphrase input via terminal. It is relevant every time a passphrase is asked. 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.

`--token-id`

Specify what token to use when unlocking the existing keyslot to get the volume key.

`--token-only`

Use only LUKS2 tokens to unlock the existing volume key.

To create a new keyslot using the passphrase provided by a token, use `--new-token-id` parameter.

`--token-type` *type*

Specify what token type (all *type* tokens) to use when unlocking the existing keyslot to get the volume key.

`--type` *type*

Specifies required device type, for more info, read the *BASIC ACTIONS* section in **cryptsetup(8)**.

`--unbound`

Creates a new LUKS2 unbound keyslot.

`--usage`

Show short option help.

`--verify-passphrase, -y`

When interactively asking for a passphrase, ask for it twice and complain if both inputs do not match. Ignored on input from file or stdin.

`--version, -V`

Show the program version.

`--volume-key-file` *file*, `--master-key-file` *file* (OBSOLETE alias)

Use a volume key stored in a file.

This allows adding a new keyslot without having to know the passphrase to the existing one. It may also be used when no keyslot is active.

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

security.

--volume-key-keyring <key description>

Use a volume key stored in a keyring. This allows one to open *luks* and *plain* device types without giving a passphrase.

For LUKS, the key and associated type have to be readable from userspace so that the volume key digest may be verified before activation. For devices in reencryption, the option may be used twice to specify both old and new volume keys.

For PLAIN type, the user must ensure that the key in the keyring is unchanged since activation. Otherwise, reloading the key can cause data corruption after an unexpected key change.

The <key description> uses keyctl-compatible syntax. This can either be a numeric key ID or a string name in the format %<key type>:<key name>. See also the **KEY IDENTIFIERS** section of **keyctl**(1). When no %<key type>: prefix is specified, we assume the key type is *user* (default type).

EXAMPLES

The interactive passphrase prompt is always the default method when not specified otherwise.

Add new keyslot using interactive passphrase prompt for both existing and new passphrases:

cryptsetup luksAddKey /dev/device

Add a new keyslot using LUKS2 tokens to unlock the existing keyslot with an interactive passphrase prompt for the new passphrase:

cryptsetup luksAddKey --token-only /dev/device

Add new keyslot using LUKS2 systemd-tpm2 tokens to unlock existing keyslot with interactive passphrase prompt for new passphrase (systemd-tpm2 token plugin must be available):

cryptsetup luksAddKey --token-type systemd-tpm2 /dev/device

Add new keyslot using interactive passphrase prompt for existing keyslot, reading new passphrase from *key_file*:

cryptsetup luksAddKey --new-keyfile key_file /dev/device or **cryptsetup luksAddKey /dev/device key_file**

Add new keyslot using volume stored in *volume_key_file* and LUKS2 token in slot 5 to get new keyslot passphrase (token in slot 5 must exist and respective token plugin must be available):

cryptsetup luksAddKey --volume-key-file volume_key_file --new-token-id 5 /dev/device

REPORTING BUGS

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

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

SEE ALSO

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

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

CRYPTSETUP

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