MyWebUniversity
python
Rocky Linux
Manual Pages
FPDF Library
PDF generator
FPDF
Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'crypttab.5'

$ man crypttab.5

CRYPTTAB(5) crypttab CRYPTTAB(5)

NAME

crypttab - Configuration for encrypted block devices

SYNOPSIS

/etc/crypttab

DESCRIPTION

The /etc/crypttab file describes encrypted block devices that are set up during system boot.

Empty lines and lines starting with the "#" character are ignored. Each of the remaining lines describes one encrypted block device. Fields are delimited by white space.

Each line is in the form

volume-name encrypted-device key-file options

The first two fields are mandatory, the remaining two are optional.

Setting up encrypted block devices using this file supports four encryption modes: LUKS, TrueCrypt, BitLocker and plain. See cryptsetup(8) for more information about each mode. When no mode is specified in the options field and the block device contains a LUKS signature, it is opened as a LUKS device; otherwise, it is assumed to be in raw dm-crypt (plain mode) format.

The four fields of /etc/crypttab are defined as follows:

1. The first field contains the name of the resulting volume with decrypted data; its block device is set up below /dev/mapper/.
2. The second field contains a path to the underlying block device or file, or a specification of a block device via "UUID=" followed by the UUID.
3. The third field specifies an absolute path to a file with the encryption key. Optionally, the path may be followed by ":" and an /etc/fstab style device specification (e.g. starting with "LABEL=" or similar); in which case the path is taken relative to the specified device's file system root. If the field is not present or is "none" or "-", a key file named after the volume to unlock (i.e. the first column of the line), suffixed with .key is automatically loaded from the /etc/cryptsetup-keys.d/ and /run/cryptsetup-keys.d/ directories, if present. Otherwise, the password has to be manually entered during system boot. For swap encryption, /dev/urandom may be used as key file, resulting in a randomized key.

   If the specified key file path refers to an AF_UNIX stream socket in the file system, the key is acquired by connecting to the socket and reading it from the connection. This allows the implementation of a service to provide key information dynamically, at the moment when it is needed. For details see below.
4. The fourth field, if present, is a comma-delimited list of options. The supported options are listed below.

KEY ACQUISITION

Six different mechanisms for acquiring the decryption key or passphrase unlocking the encrypted volume are supported. Specifically:

1. Most prominently, the user may be queried interactively during volume activation (i.e. typically at boot), asking them to type in the necessary passphrases.
2. The (unencrypted) key may be read from a file on disk, possibly on

removable media. The third field of each line encodes the location, for details see above.

3. The (unencrypted) key may be requested from another service, by specifying an AF_UNIX file system socket in place of a key file in the third field. For details see above and below.
4. The key may be acquired via a PKCS#11 compatible hardware security token or smartcard. In this case an encrypted key is stored on disk/removable media, acquired via AF_UNIX, or stored in the LUKS2 JSON token metadata header. The encrypted key is then decrypted by the PKCS#11 token with an RSA key stored on it, and then used to unlock the encrypted volume. Use the pkcs11-uri= option described below to use this mechanism.
5. Similarly, the key may be acquired via a FIDO2 compatible hardware security token (which must implement the "hmac-secret" extension). In this case a key generated randomly during enrollment is stored on disk/removable media, acquired via AF_UNIX, or stored in the LUKS2 JSON token metadata header. The random key is hashed via a keyed hash function (HMAC) on the FIDO2 token, using a secret key stored on the token that never leaves it. The resulting hash value is then used as key to unlock the encrypted volume. Use the fido2-device= option described below to use this mechanism.
6. Similarly, the key may be acquired via a TPM2 security chip. In this case a (during enrollment) randomly generated key ? encrypted by an asymmetric key derived from the TPM2 chip's seed key ? is stored on disk/removable media, acquired via AF_UNIX, or stored in the LUKS2 JSON token metadata header. Use the tpm2-device= option described below to use this mechanism.

For the latter five mechanisms the source for the key material used for unlocking the volume is primarily configured in the third field of each /etc/crypttab line, but may also configured in /etc/cryptsetup-keys.d/ and /run/cryptsetup-keys.d/ (see above) or in the LUKS2 JSON token header (in case of the latter three). Use the systemd-cryptenroll(1) tool to enroll PKCS#11, FIDO2 and TPM2 devices in LUKS2 volumes.

SUPPORTED OPTIONS

The following options may be used in the fourth field of each line:

cipher=

Specifies the cipher to use. See cryptsetup(8) for possible values and the default value of this option. A cipher with unpredictable IV values, such as "aes-cbc-essiv:sha256", is recommended. Embedded commas in the cipher specification need to be escaped by preceding them with a backslash, see example below.

discard

Allow discard requests to be passed through the encrypted block device. This improves performance on SSD storage but has security implications.

hash=

Specifies the hash to use for password hashing. See cryptsetup(8) for possible values and the default value of this option.

header=

Use a detached (separated) metadata device or file where the LUKS header is stored. This option is only relevant for LUKS devices. See cryptsetup(8) for possible values and the default value of this option.

Optionally, the path may be followed by ":" and an /etc/fstab device specification (e.g. starting with "UUID=" or similar); in which case, the path is relative to the device file system root. The device gets mounted automatically for LUKS device activation duration only.

keyfile-offset=

Specifies the number of bytes to skip at the start of the key file. See cryptsetup(8) for possible values and the default value of this option.

keyfile-size=

Specifies the maximum number of bytes to read from the key file. See cryptsetup(8) for possible values and the default value of this option. This option is ignored in plain encryption mode, as the key file size is then given by the key size.

keyfile-erase

If enabled, the specified key file is erased after the volume is activated or when activation fails. This is in particular useful when the key file is only acquired transiently before activation (e.g. via a file in /run/, generated by a service running before activation), and shall be removed after use. Defaults to off.

key-slot=

Specifies the key slot to compare the passphrase or key against. If the key slot does not match the given passphrase or key, but another would, the setup of the device will fail regardless. This option implies luks. See cryptsetup(8) for possible values. The default is to try all key slots in sequential order.

keyfile-timeout=

Specifies the timeout for the device on which the key file resides or the device used as the key file, and falls back to a password if it could not be accessed. See systemd-cryptsetup-generator(8) for key files on external devices.

luks

Force LUKS mode. When this mode is used, the following options are ignored since they are provided by the LUKS header on the device: cipher=, hash=, size=.

bitlk

Decrypt BitLocker drive. Encryption parameters are deduced by cryptsetup from BitLocker header.

_netdev

Marks this cryptsetup device as requiring network. It will be started after the network is available, similarly to systemd.mount(5) units marked with _netdev. The service unit to set up this device will be ordered between remote-fs-pre.target and remote-cryptsetup.target, instead of cryptsetup-pre.target and cryptsetup.target.

Hint: if this device is used for a mount point that is specified in fstab(5), the _netdev option should also be used for the mount point. Otherwise, a dependency loop might be created where the mount point will be pulled in by local-fs.target, while the service to configure the network is usually only started after the local file system has been mounted.

noauto

This device will not be added to cryptsetup.target. This means that it will not be automatically unlocked on boot, unless something else pulls it in. In particular, if the device is used for a mount point, it'll be unlocked automatically during boot, unless the mount point itself is also disabled with noauto.

nofail

This device will not be a hard dependency of cryptsetup.target. It'll still be pulled in and started, but the system will not wait for the device to show up and be unlocked, and boot will not fail if this is unsuccessful. Note that other units that depend on the unlocked device may still fail. In particular, if the device is used for a mount point, the mount point itself also needs to have the nofail option, or the boot will fail if the device is not unlocked successfully.

offset=

Start offset in the backend device, in 512-byte sectors. This option is only relevant for plain devices.

plain

Force plain encryption mode.

read-only, readonly

Set up the encrypted block device in read-only mode.

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 requires kernel 4.0 or newer.

submit-from-crypt-cpus

Disable offloading writes to a separate thread after encryption.

There are some situations where offloading write requests from the encryption threads to a dedicated thread degrades performance significantly. The default is to offload write requests to a dedicated thread because it benefits the CFQ scheduler to have writes submitted using the same context.

This requires kernel 4.0 or newer.

no-read-workqueue

Bypass dm-crypt internal workqueue and process read requests synchronously. The default is to queue these requests and process them asynchronously.

This requires kernel 5.9 or newer.

no-write-workqueue

Bypass dm-crypt internal workqueue and process write requests synchronously. The default is to queue these requests and process them asynchronously.

This requires kernel 5.9 or newer.

skip=

How many 512-byte sectors of the encrypted data to skip at the beginning. This is different from the offset= option with respect to the sector numbers used in initialization vector (IV) calculation. Using offset= will shift the IV calculation by the same negative amount. Hence, if offset=n is given, sector n will get a sector number of 0 for the IV calculation. Using skip= causes sector n to also be the first sector of the mapped device, but with its number for IV generation being n.

This option is only relevant for plain devices.

size=

Specifies the key size in bits. See cryptsetup(8) for possible values and the default value of this option.

sector-size=

Specifies the sector size in bytes. See cryptsetup(8) for possible values and the default value of this option.

swap

The encrypted block device will be used as a swap device, and will be formatted accordingly after setting up the encrypted block device, with mkswap(8). This option implies plain.

WARNING: Using the swap option will destroy the contents of the named partition during every boot, so make sure the underlying block device is specified correctly.

tcrypt

Use TrueCrypt encryption mode. When this mode is used, the following options are ignored since they are provided by the TrueCrypt header on the device or do not apply: cipher=, hash=, keyfile-offset=, keyfile-size=, size=.

When this mode is used, the passphrase is read from the key file given in the third field. Only the first line of this file is read, excluding the new line character.

Note that the TrueCrypt format uses both passphrase and key files to derive a password for the volume. Therefore, the passphrase and all key files need to be provided. Use tcrypt-keyfile= to provide the absolute path to all key files. When using an empty passphrase in combination with one or more key files, use "/dev/null" as the password file in the third field.

tcrypt-hidden

Use the hidden TrueCrypt volume. This option implies tcrypt.

This will map the hidden volume that is inside of the volume provided in the second field. Please note that there is no protection for the hidden volume if the outer volume is mounted instead. See cryptsetup(8) for more information on this limitation.

tcrypt-keyfile=

Specifies the absolute path to a key file to use for a TrueCrypt volume. This implies tcrypt and can be used more than once to provide several key files.

See the entry for tcrypt on the behavior of the passphrase and key files when using TrueCrypt encryption mode.

tcrypt-system

Use TrueCrypt in system encryption mode. This option implies tcrypt.

tcrypt-veracrypt

Check for a VeraCrypt volume. VeraCrypt is a fork of TrueCrypt that is mostly compatible, but uses different, stronger key derivation algorithms that cannot be detected without this flag. Enabling this option could substantially slow down unlocking, because VeraCrypt's key derivation takes much longer than TrueCrypt's. This option implies tcrypt.

timeout=

Specifies the timeout for querying for a password. If no unit is specified, seconds is used. Supported units are s, ms, us, min, h, d. A timeout of 0 waits indefinitely (which is the default).

tmp=

The encrypted block device will be prepared for using it as /tmp/; it will be formatted using mkfs(8). Takes a file system type as argument, such as "ext4", "xfs" or "btrfs". If no argument is specified defaults to "ext4". This option implies plain.

WARNING: Using the tmp option will destroy the contents of the named partition during every boot, so make sure the underlying block device is specified correctly.

tries=

Specifies the maximum number of times the user is queried for a password. The default is 3. If set to 0, the user is queried for a password indefinitely.

headless=

Takes a boolean argument, defaults to false. If true, never query interactively for the password/PIN. Useful for headless systems.

verify

If the encryption password is read from console, it has to be entered twice to prevent typos.

password-echo=yes|no|masked

Controls whether to echo passwords or security token PINs that are read from console. Takes a boolean or the special string "masked". The default is password-echo=masked.

If enabled, the typed characters are echoed literally. If disabled, the typed characters are not echoed in any form, the user will not get feedback on their input. If set to "masked", an asterisk ("*") is echoed for each character typed. Regardless of which mode is chosen, if the user hits the tabulator key ("?") at any time, or the backspace key ("?") before any other data has been entered, then echo is turned off.

pkcs11-uri=

Takes either the special value "auto" or an RFC7512 PKCS#11 URI[1] pointing to a private RSA key which is used to decrypt the encrypted key specified in the third column of the line. This is useful for unlocking encrypted volumes through PKCS#11 compatible security tokens or smartcards. See below for an example how to set up this mechanism for unlocking a LUKS2 volume with a YubiKey security token.

If specified as "auto" the volume must be of type LUKS2 and must carry PKCS#11 security token metadata in its LUKS2 JSON token section. In this mode the URI and the encrypted key are automatically read from the LUKS2 JSON token header. Use systemd-cryptenroll(1) as simple tool for enrolling PKCS#11 security tokens or smartcards in a way compatible with "auto". In this mode the third column of the line should remain empty (that is, specified as "-").

The specified URI can refer directly to a private RSA key stored on a token or alternatively just to a slot or token, in which case a search for a suitable private RSA key will be performed. In this case if multiple suitable objects are found the token is refused.

The encrypted key configured in the third column of the line is passed as is (i.e. in binary form, unprocessed) to RSA decryption. The resulting decrypted key is then Base64 encoded before it is used to unlock the LUKS volume.

Use systemd-cryptenroll --pkcs11-token-uri=list to list all suitable PKCS#11 security tokens currently plugged in, along with their URIs.

Note that many newer security tokens that may be used as PKCS#11 security token typically also implement the newer and simpler FIDO2 standard. Consider using fido2-device= (described below) to enroll it via FIDO2 instead. Note that a security token enrolled via PKCS#11 cannot be used to unlock the volume via FIDO2, unless also enrolled via FIDO2, and vice versa.

fido2-device=

Takes either the special value "auto" or the path to a "hidraw" device node (e.g. /dev/hidraw1) referring to a FIDO2 security token that implements the "hmac-secret" extension (most current hardware security tokens do). See below for an example how to set up this mechanism for unlocking an encrypted volume with a FIDO2 security token.

If specified as "auto" the FIDO2 token device is automatically discovered, as it is plugged in.

FIDO2 volume unlocking requires a client ID hash (CID) to be configured via fido2-cid= (see below) and a key to pass to the security token's HMAC functionality (configured in the line's third column) to operate. If not configured and the volume is of type LUKS2, the CID and the key are read from LUKS2 JSON token metadata instead. Use systemd-cryptenroll(1) as simple tool for enrolling FIDO2 security tokens, compatible with this automatic mode, which is only available for LUKS2 volumes.

Use systemd-cryptenroll --fido2-device=list to list all suitable FIDO2 security tokens currently plugged in, along with their device nodes.

This option implements the following mechanism: the configured key is hashed via they HMAC keyed hash function the FIDO2 device implements, keyed by a secret key embedded on the device. The resulting hash value is Base64 encoded and used to unlock the LUKS2 volume. As it should not be possible to extract the secret from the hardware token, it should not be possible to retrieve the hashed key given the configured key ? without possessing the hardware token.

Note that many security tokens that implement FIDO2 also implement PKCS#11, suitable for unlocking volumes via the pkcs11-uri= option described above. Typically the newer, simpler FIDO2 standard is preferable.

fido2-cid=

Takes a Base64 encoded FIDO2 client ID to use for the FIDO2 unlock operation. If specified, but fido2-device= is not, fido2-device=auto is implied. If fido2-device= is used but fido2-cid= is not, the volume must be of LUKS2 type, and the CID is read from the LUKS2 JSON token header. Use systemd-cryptenroll(1) for enrolling a FIDO2 token in the LUKS2 header compatible with this automatic mode.

fido2-rp=

Takes a string, configuring the FIDO2 Relying Party (rp) for the FIDO2 unlock operation. If not specified "io.systemd.cryptsetup" is used, except if the LUKS2 JSON token header contains a different value. It should normally not be necessary to override this.

tpm2-device=

Takes either the special value "auto" or the path to a device node (e.g. /dev/tpmrm0) referring to a TPM2 security chip. See below for an example how to set up this mechanism for unlocking an encrypted volume with a TPM2 chip.

Use tpm2-pcrs= (see below) to configure the set of TPM2 PCRs to bind the volume unlocking to. Use systemd-cryptenroll(1) as simple tool for enrolling TPM2 security chips in LUKS2 volumes.

If specified as "auto" the TPM2 device is automatically discovered. Use systemd-cryptenroll --tpm2-device=list to list all suitable TPM2 devices currently available, along with their device nodes.

This option implements the following mechanism: when enrolling a TPM2 device via systemd-cryptenroll on a LUKS2 volume, a randomized key unlocking the volume is generated on the host and loaded into the TPM2 chip where it is encrypted with an asymmetric "primary" key pair derived from the TPM2's internal "seed" key. Neither the seed key nor the primary key are permitted to ever leave the TPM2 chip ? however, the now encrypted randomized key may. It is saved in the LUKS2 volume JSON token header. When unlocking the encrypted volume, the primary key pair is generated on the TPM2 chip again (which works as long as the chip's seed key is correctly maintained by the TPM2 chip), which is then used to decrypt (on the TPM2 chip) the encrypted key from the LUKS2 volume JSON token header saved there during enrollment. The resulting decrypted key is then used to unlock the volume. When the randomized key is encrypted the current values of the selected PCRs (see below) are included in the operation, so that different PCR state results in different encrypted keys and the decrypted key can only be recovered if the same PCR state is reproduced.

tpm2-pcrs=

Takes a "+" separated list of numeric TPM2 PCR (i.e. "Platform Configuration Register") indexes to bind the TPM2 volume unlocking to. This option is only useful when TPM2 enrollment metadata is not available in the LUKS2 JSON token header already, the way systemd-cryptenroll writes it there. If not used (and no metadata in the LUKS2 JSON token header defines it), defaults to a list of a single entry: PCR 7. Assign an empty string to encode a policy that binds the key to no PCRs, making the key accessible to local programs regardless of the current PCR state.

tpm2-pin=

Takes a boolean argument, defaults to "false". Controls whether TPM2 volume unlocking is bound to a PIN in addition to PCRs. Similarly, this option is only useful when TPM2 enrollment metadata is not available.

tpm2-signature=

Takes an absolute path to a TPM2 PCR JSON signature file, as produced by the systemd-measure(1) tool. This permits locking LUKS2 volumes to any PCR values for which a valid signature matching a public key specified at key enrollment time can be provided. See systemd-cryptenroll(1) for details on enrolling TPM2 PCR public keys. If this option is not specified but it is attempted to unlock a LUKS2 volume with a signed TPM2 PCR enrollment a suitable signature file tpm2-pcr-signature.json is searched for in /etc/systemd/, /run/systemd/, /usr/lib/systemd/ (in this order).

token-timeout=

Specifies how long to wait at most for configured security devices (i.e. FIDO2, PKCS#11, TPM2) to show up. Takes a time value in seconds (but other time units may be specified too, see systemd.time(7) for supported formats). Defaults to 30s. Once the specified timeout elapsed authentication via password is attempted. Note that this timeout applies to waiting for the security device to show up ? it does not apply to the PIN prompt for the device (should one be needed) or similar. Pass 0 to turn off the time-out and wait forever.

try-empty-password=

Takes a boolean argument. If enabled, right before asking the user for a password it is first attempted to unlock the volume with an empty password. This is useful for systems that are initialized with an encrypted volume with only an empty password set, which shall be replaced with a suitable password during first boot, but after activation.

x-systemd.device-timeout=

Specifies how long systemd should wait for a block device to show up before giving up on the entry. The argument is a time in seconds or explicitly specified units of "s", "min", "h", "ms".

x-initrd.attach

Setup this encrypted block device in the initrd, similarly to systemd.mount(5) units marked with x-initrd.mount.

Although it's not necessary to mark the mount entry for the root file system with x-initrd.mount, x-initrd.attach is still recommended with the encrypted block device containing the root file system as otherwise systemd will attempt to detach the device during the regular system shutdown while it's still in use. With this option the device will still be detached but later after the root file system is unmounted.

All other encrypted block devices that contain file systems mounted in the initrd should use this option.

At early boot and when the system manager configuration is reloaded, this file is translated into native systemd units by systemd-cryptsetup-generator(8).

AF_UNIX KEY FILES

If the key file path (as specified in the third column of /etc/crypttab entries, see above) refers to an AF_UNIX stream socket in the file system, the key is acquired by connecting to the socket and reading the key from the connection. The connection is made from an AF_UNIX socket name in the abstract namespace, see unix(7) for details. The source socket name is chosen according the following format:

```
NUL RANDOM /cryptsetup/ VOLUME
```

In other words: a NUL byte (as required for abstract namespace sockets), followed by a random string (consisting of alphanumeric characters only), followed by the literal string "/cryptsetup/", followed by the name of the volume to acquire they key for. For example, for the volume "myvol":

```
\0d7067f78d9827418/cryptsetup/myvol
```

Services listening on the AF_UNIX stream socket may query the source socket name with getpeername(2), and use this to determine which key to send, allowing a single listening socket to serve keys for multiple volumes. If the PKCS#11 logic is used (see above), the socket source name is picked in similar fashion, except that the literal string "/cryptsetup-pkcs11/" is used. And similarly for FIDO2

("/cryptsetup-fido2/") and TPM2 ("/cryptsetup-tpm2/"). A diffent path component is used so that services providing key material know that the secret key was not requested directly, but instead an encrypted key that will be decrypted via the PKCS#11/FIDO2/TPM2 logic to acquire the final secret key.

EXAMPLES

Example 1. /etc/crypttab example

Set up four encrypted block devices. One using LUKS for normal storage, another one for usage as a swap device and two TrueCrypt volumes. For the fourth device, the option string is interpreted as two options "cipher=xchacha12,aes-adiantum-plain64", "keyfile-timeout=10s".

```
luks       UUID=2505567a-9e27-4efe-a4d5-15ad146c258b
swap       /dev/sda7       /dev/urandom       swap
truecrypt  /dev/sda2       /etc/container_password  tcrypt
hidden     /mnt/tc_hidden  /dev/null    tcrypt-hidden,tcrypt-keyfile=/etc/keyfile
external   /dev/sda3       keyfile:LABEL=keydev keyfile-timeout=10s,cipher=xchacha12\,aes-adiantum-plain64
```

Example 2. Yubikey-based PKCS#11 Volume Unlocking Example

The PKCS#11 logic allows hooking up any compatible security token that is capable of storing RSA decryption keys for unlocking an encrypted volume. Here's an example how to set up a Yubikey security token for this purpose on a LUKS2 volume, using ykmap(1) from the yubikey-manager project to initialize the token and systemd-cryptenroll(1) to add it in the LUKS2 volume:

```
# SPDX-License-Identifier: MIT-0
# Destroy any old key on the Yubikey (careful!)
ykman piv reset
# Generate a new private/public key pair on the device, store the public key in
# 'pubkey.pem'.
ykman piv generate-key -a RSA2048 9d pubkey.pem
# Create a self-signed certificate from this public key, and store it on the
# device. The "subject" should be an arbitrary user-chosen string to identify
# the token with.
ykman piv generate-certificate --subject "Knobelei" 9d pubkey.pem
```

```
    # We don't need the public key anymore, let's remove it. Since it is not
    # security sensitive we just do a regular "rm" here.
    rm pubkey.pem
    # Enroll the freshly initialized security token in the LUKS2 volume. Replace
    # /dev/sdXn by the partition to use (e.g. /dev/sda1).
    sudo systemd-cryptenroll --pkcs11-token-uri=auto /dev/sdXn
    # Test: Let's run systemd-cryptsetup to test if this all worked.
    sudo /usr/lib/systemd/systemd-cryptsetup attach mytest /dev/sdXn - pkcs11-uri=auto
    # If that worked, let's now add the same line persistently to /etc/crypttab,
    # for the future.
    sudo bash -c 'echo "mytest /dev/sdXn - pkcs11-uri=auto" >> /etc/crypttab'
```

A few notes on the above:

? We use RSA2048, which is the longest key size current Yubikeys support

? We use Yubikey key slot 9d, since that's apparently the keyslot to use for decryption purposes, see documentation[2].

Example 3. FIDO2 Volume Unlocking Example

The FIDO2 logic allows using any compatible FIDO2 security token that implements the "hmac-secret" extension for unlocking an encrypted volume. Here's an example how to set up a FIDO2 security token for this purpose for a LUKS2 volume, using systemd-cryptenroll(1):

```
    # SPDX-License-Identifier: MIT-0
    # Enroll the security token in the LUKS2 volume. Replace /dev/sdXn by the
    # partition to use (e.g. /dev/sda1).
    sudo systemd-cryptenroll --fido2-device=auto /dev/sdXn
    # Test: Let's run systemd-cryptsetup to test if this worked.
    sudo /usr/lib/systemd/systemd-cryptsetup attach mytest /dev/sdXn - fido2-device=auto
    # If that worked, let's now add the same line persistently to /etc/crypttab,
    # for the future.
    sudo bash -c 'echo "mytest /dev/sdXn - fido2-device=auto" >> /etc/crypttab'
```

Example 4. TPM2 Volume Unlocking Example

The TPM2 logic allows using any TPM2 chip supported by the Linux kernel for unlocking an encrypted volume. Here's an example how to set up a

TPM2 chip for this purpose for a LUKS2 volume, using systemd-cryptenroll(1):

```
# SPDX-License-Identifier: MIT-0

# Enroll the TPM2 security chip in the LUKS2 volume, and bind it to PCR 7

# only. Replace /dev/sdXn by the partition to use (e.g. /dev/sda1).

sudo systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=7 /dev/sdXn

# Test: Let's run systemd-cryptsetup to test if this worked.

sudo /usr/lib/systemd/systemd-cryptsetup attach mytest /dev/sdXn - tpm2-device=auto

# If that worked, let's now add the same line persistently to /etc/crypttab,

# for the future.

sudo bash -c 'echo "mytest /dev/sdXn - tpm2-device=auto" >> /etc/crypttab'
```

SEE ALSO

systemd(1), systemd-cryptsetup@.service(8), systemd-cryptsetup-generator(8), systemd-cryptenroll(1), fstab(5), cryptsetup(8), mkswap(8), mke2fs(8)

NOTES

1. RFC7512 PKCS#11 URI

   https://tools.ietf.org/html/rfc7512

2. see documentation

   https://developers.yubico.com/PIV/Introduction/Certificate_slots.html