Name

systemd-cryptenroll — Enroll PKCS#11, FIDO2, TPM2 token/devices to LUKS2 encrypted volumes

Synopsis

`systemd-cryptenroll` [OPTIONS...] [DEVICE]

Description

systemd-cryptenroll is a tool for enrolling hardware security tokens and devices into a LUKS2 encrypted volume, which may then be used to unlock the volume during boot. Specifically, it supports tokens and credentials of the following kind to be enrolled:

1. PKCS#11 security tokens and smartcards that may carry an RSA or EC key pair (e.g. various YubiKeys)
2. FIDO2 security tokens that implement the "`hmac-secret`" extension (most FIDO2 keys, including YubiKeys)
3. TPM2 security devices
4. Regular passphrases
5. Recovery keys. These are similar to regular passphrases, however are randomly generated on the computer and thus generally have higher entropy than user-chosen passphrases. Their character set has been designed to ensure they are easy to type in, while having high entropy. They may also be scanned off screen using QR codes. Recovery keys may be used for unlocking LUKS2 volumes wherever passphrases are accepted. They are intended to be used in combination with an enrolled hardware security token, as a recovery option when the token is lost.

In addition, the tool may be used to enumerate currently enrolled security tokens and wipe a subset of them. The latter may be combined with the enrollment operation of a new security token, in order to update or replace enrollments.

The tool supports only LUKS2 volumes, as it stores token meta-information in the LUKS2 JSON token area, which is not available in other encryption formats.

systemd-cryptsetup operates on the device backing `/var/` if no device is specified explicitly, and no wipe operation is requested. (Note that in the typical case where `/var/` is on the same file system as the root file system, this hence enrolls a key into the backing device of the root file system.)

TPM2 PCRs and policies

PCRs allow binding of the encryption of secrets to specific software versions and system state, so that the enrolled key is only accessible (may be "unsealed") if specific trusted software and/or configuration is used. Such bindings may be created with the option `--tpm2-pcrs=` described below.

Secrets may also be bound indirectly: a signed policy for a state of some combination of PCR values is provided, and the secret is bound to the public part of the key used to sign this policy. This means that the owner of a key can generate a sequence of signed policies, for specific software versions and system states, and the secret can be decrypted as long as the machine state matches one of those policies. For example, a vendor may provide such a policy for each kernel+initrd update, allowing users to encrypt secrets so that they can be decrypted when running any kernel+initrd signed by the vendor. Such bindings may be created with the options `--tpm2-public-key=`, `--tpm2-public-key-pcrs=`, `--tpm2-signature=` described below.

See Linux TPM PCR Registry for an authoritative list of PCRs and how they are updated. The table below contains a quick reference, describing in particular the PCRs modified by systemd.

Table 1. Well-known PCR Definitions

PCR	name	Explanation
0	platform-code	Core system firmware executable code; changes on firmware updates
1	platform-config	Core system firmware data/host platform configuration; typically contains serial and model numbers, changes on basic hardware/CPU/RAM replacements
2	external-code	Extended or pluggable executable code; includes option ROMs on pluggable hardware
3	external-config	Extended or pluggable firmware data; includes information about pluggable hardware
4	boot-loader-code	Boot loader and additional drivers, PE binaries invoked by the boot loader; changes on boot loader updates. sd-stub(7) measures system extension images read from the ESP here too (see systemd-sysext(8)).
5	boot-loader-config	GPT/Partition table; changes when the partitions are added, modified, or removed
7	secure-boot-policy	Secure Boot state; changes when UEFI SecureBoot mode is enabled/disabled, or firmware certificates (PK, KEK, db, dbx, …) changes.
9	kernel-initrd	The Linux kernel measures all initrds it receives into this PCR.
10	ima	The IMA project measures its runtime state into this PCR.
11	kernel-boot	systemd-stub(7) measures the ELF kernel image, embedded initrd and other payload of the PE image it is placed in into this PCR. systemd-pcrphase.service(8) measures boot phase strings into this PCR at various milestones of the boot process.
12	kernel-config	systemd-boot(7) measures the kernel command line into this PCR. systemd-stub(7) measures any manually specified kernel command line (i.e. a kernel command line that overrides the one embedded in the unified PE image) and loaded credentials into this PCR.
13	sysexts	systemd-stub(7) measures any systemd-sysext(8) images it passes to the booted kernel into this PCR.
14	shim-policy	The shim project measures its "MOK" certificates and hashes into this PCR.
15	system-identity	systemd-cryptsetup(8) optionally measures the volume key of activated LUKS volumes into this PCR. systemd-pcrmachine.service(8) measures the machine-id(5) into this PCR. systemd-pcrfs@.service(8) measures mount points, file system UUIDs, labels, partition UUIDs of the root and `/var/` filesystems into this PCR.
16	debug	Debug
23	application-support	Application Support

In general, encrypted volumes would be bound to some combination of PCRs 7, 11, and 14 (if shim/MOK is used). In order to allow firmware and OS version updates, it is typically not advisable to use PCRs such as 0 and 2, since the program code they cover should already be covered indirectly through the certificates measured into PCR 7. Validation through certificates hashes is typically preferable over validation through direct measurements as it is less brittle in context of OS/firmware updates: the measurements will change on every update, but signatures should remain unchanged. See the Linux TPM PCR Registry for more discussion.

Limitations

Note that currently when enrolling a new key of one of the five supported types listed above, it is required to first provide a passphrase, a recovery key, a FIDO2 token, or a TPM2 key. It's currently not supported to unlock a device with a PKCS#11 key in order to enroll a new PKCS#11 key. Thus, if in future key roll-over is desired it's generally recommended to ensure a passphrase, a recovery key, a FIDO2 token, or a TPM2 key is always enrolled.

Also note that support for enrolling multiple FIDO2 tokens is currently limited. When multiple FIDO2 tokens are enrolled, **systemd-cryptsetup** will perform pre-flight requests to attempt to identify which of the enrolled tokens are currently plugged in. However, this is not possible for FIDO2 tokens with user verification (UV, usually via biometrics), in which case it will fall back to attempting each enrolled token one by one. This will result in multiple prompts for PIN and user verification. This limitation does not apply to PKCS#11 tokens.

Compatibility

Security technology both in systemd and in the general industry constantly evolves. In order to provide best security guarantees, the way TPM2, FIDO2, PKCS#11 devices are enrolled is regularly updated in newer versions of systemd. Whenever this happens the following compatibility guarantees are given:

- Old enrollments continue to be supported and may be unlocked with newer versions of systemd-cryptsetup@.service(8).
- The opposite is not guaranteed however: it might not be possible to unlock volumes with enrollments done with a newer version of **systemd-cryptenroll** with an older version of **systemd-cryptsetup**.

That said, it is generally recommended to use matching versions of **systemd-cryptenroll** and **systemd-cryptsetup**, since this is best tested and supported.

It might be advisable to re-enroll existing enrollments to take benefit of newer security features, as they are added to systemd.

Unlocking

The following options are understood that may be used to unlock the device in preparation of the enrollment operations:

`--unlock-key-file=`*`PATH`*

> Use a file instead of a password/passphrase read from stdin to unlock the volume. Expects the PATH to the file containing your key to unlock the volume. Currently there is nothing like `--key-file-offset=` or `--key-file-size=` so this file has to only contain the full key.
>
> Added in version 252.

`--unlock-fido2-device=`*`PATH`*

> Use a FIDO2 device instead of a password/passphrase read from stdin to unlock the volume. Expects a `hidraw` device referring to the FIDO2 device (e.g. `/dev/hidraw1`). Alternatively the special value "`auto`" may be specified, in order to automatically determine the device node of a currently plugged in security token (of which there must be exactly one). This automatic discovery is unsupported if `--fido2-device=` option is also specified. Note that currently FIDO2 devices enrolled without an accompanying LUKS2 token (i.e. `--fido2-parameters-in-header=no`) cannot be used for unlocking.
>
> Added in version 253.

`--unlock-tpm2-device=`*`PATH`*

Use a TPM2 device instead of a password/passphrase read from stdin to unlock the volume. Expects a device node path referring to the TPM2 chip (e.g. `/dev/tpmrm0`). Alternatively the special value "`auto`" may be specified, in order to automatically determine the device node of a currently discovered TPM2 device (of which there must be exactly one).

Added in version 256.

Simple Enrollment

The following options are understood that may be used to enroll simple user input based unlocking:

`--password`

Enroll a regular password/passphrase. This command is mostly equivalent to **cryptsetup luksAddKey**, however may be combined with `--wipe-slot=` in one call, see below.

Added in version 248.

`--recovery-key`

Enroll a recovery key. Recovery keys are mostly identical to passphrases, but are computer-generated instead of being chosen by a human, and thus have a guaranteed high entropy. The key uses a character set that is easy to type in, and may be scanned off screen via a QR code.

Added in version 248.

PKCS#11 Enrollment

The following option is understood that may be used to enroll PKCS#11 tokens:

`--pkcs11-token-uri=`*`URI`*

Enroll a PKCS#11 security token or smartcard (e.g. a YubiKey). Expects a PKCS#11 URI that allows finding an X.509 certificate or a public key on the token. The URI must also be suitable to find a related private key after changing the type of object in it. Alternatively the special value "`auto`" may be specified, in order to automatically determine the suitable URI if a single security token containing a single key pair is plugged in. The special value "`list`" may be used to enumerate all suitable PKCS#11 tokens currently plugged in.

The PKCS#11 token must contain an RSA or EC key pair which will be used to unlock a LUKS2 volume. For RSA, a randomly generated volume key is encrypted with a public key in the token, and stored in the LUKS2 JSON token header area. To unlock a volume, the stored encrypted volume key will be decrypted with a private key in the token. For ECC, ECDH algorithm is used: we generate a pair of EC keys in the same EC group, then derive a shared secret using the generated private key and the public key in the token. The derived shared secret is used as a volume key. The generated public key is stored in the LUKS2 JSON token header area. The generated private key is erased. To unlock a volume, we derive the shared secret with the stored public key and a private key in the token.

In order to unlock a LUKS2 volume with an enrolled PKCS#11 security token, specify the `pkcs11-uri=` option in the respective `/etc/crypttab` line:

```
myvolume /dev/sda1 - pkcs11-uri=auto
```

See crypttab(5) for a more comprehensive example of a **systemd-cryptenroll** invocation and its matching `/etc/crypttab` line.

Added in version 248.

FIDO2 Enrollment

The following options are understood that may be used to enroll PKCS#11 tokens:

`--fido2-credential-algorithm=`*`STRING`*

> Specify COSE algorithm used in credential generation. The default value is "`es256`". Supported values are "`es256`", "`rs256`" and "`eddsa`".
>
> "`es256`" denotes ECDSA over NIST P-256 with SHA-256. "`rs256`" denotes 2048-bit RSA with PKCS#1.5 padding and SHA-256. "`eddsa`" denotes EDDSA over Curve25519 with SHA-512.
>
> Note that your authenticator may choose not to support some algorithms.
>
> Added in version 251.

`--fido2-device=`*`PATH`*

> Enroll a FIDO2 security token that implements the "`hmac-secret`" extension (e.g. a YubiKey). Expects a `hidraw` device referring to the FIDO2 device (e.g. `/dev/hidraw1`). Alternatively the special value "`auto`" may be specified, in order to automatically determine the device node of a currently plugged in security token (of which there must be exactly one). This automatic discovery is unsupported if `--unlock-fido2-device=` option is also specified. The special value "`list`" may be used to enumerate all suitable FIDO2 tokens currently plugged in. Note that many hardware security tokens that implement FIDO2 also implement the older PKCS#11 standard. Typically FIDO2 is preferable, given it's simpler to use and more modern.
>
> In order to unlock a LUKS2 volume with an enrolled FIDO2 security token, specify the `fido2-device=` option in the respective `/etc/crypttab` line:
>
> ```
> myvolume /dev/sda1 - fido2-device=auto
> ```
>
> See crypttab(5) for a more comprehensive example of a **systemd-cryptenroll** invocation and its matching `/etc/crypttab` line.
>
> Added in version 248.

`--fido2-salt-file=`*`PATH`*

> When enrolling a FIDO2 security token, specifies the path to a file or an `AF_UNIX` socket from which we should read the salt value to be used in the HMAC operation performed by the FIDO2 security token. If this option is not specified, the salt will be randomly generated.
>
> Added in version 257.

`--fido2-parameters-in-header=`*`BOOL`*

> When enrolling a FIDO2 security token, controls whether to store FIDO2 parameters in a token in the LUKS2 superblock. Defaults to "`yes`". If set to "`no`", the `fido2-cid=` option has to be specified manually in the respective `/etc/crypttab` line along with a key file. See crypttab(5) for details.
>
> Added in version 257.

`--fido2-with-client-pin=`*`BOOL`*

> When enrolling a FIDO2 security token, controls whether to require the user to enter a PIN when unlocking the volume (the FIDO2 "`clientPin`" feature). Defaults to "`yes`". (Note: this setting is without effect if the security token does not support the "`clientPin`" feature at all, or does not allow enabling or disabling it.)

Added in version 249.

`--fido2-with-user-presence=`*`BOOL`*

When enrolling a FIDO2 security token, controls whether to require the user to verify presence (tap the token, the FIDO2 "`up`" feature) when unlocking the volume. Defaults to "`yes`". (Note: this setting is without effect if the security token does not support the "`up`" feature at all, or does not allow enabling or disabling it.)

Added in version 249.

`--fido2-with-user-verification=`*`BOOL`*

When enrolling a FIDO2 security token, controls whether to require user verification when unlocking the volume (the FIDO2 "`uv`" feature). Defaults to "`no`". (Note: this setting is without effect if the security token does not support the "`uv`" feature at all, or does not allow enabling or disabling it.)

Added in version 249.

TPM2 Enrollment

The following options are understood that may be used to enroll TPM2 devices:

`--tpm2-device=`*`PATH`*

Enroll a TPM2 security chip. Expects a device node path referring to the TPM2 chip (e.g. `/dev/tpmrm0`). Alternatively the special value "`auto`" may be specified, in order to automatically determine the device node of a currently discovered TPM2 device (of which there must be exactly one). The special value "`list`" may be used to enumerate all suitable TPM2 devices currently discovered.

In order to unlock a LUKS2 volume with an enrolled TPM2 security chip, specify the `tpm2-device=` option in the respective `/etc/crypttab` line:

```
myvolume /dev/sda1 - tpm2-device=auto
```

See crypttab(5) for a more comprehensive example of a **systemd-cryptenroll** invocation and its matching `/etc/crypttab` line.

Use `--tpm2-pcrs=` (see below) to configure which TPM2 PCR indexes to bind the enrollment to.

Added in version 248.

`--tpm2-device-key=`*`PATH`*

Enroll a TPM2 security chip using its public key. Expects a path referring to the TPM2 public key in TPM2B_PUBLIC format. This cannot be used with `--tpm2-device=`, as it performs the same operation, but without connecting to the TPM2 security chip; instead the enrollment is calculated using the provided TPM2 key. This is useful in situations where the TPM2 security chip is not available at the time of enrollment.

The key, in most cases, should be the Storage Root Key (SRK) from a local TPM2 security chip. If a key from a different handle (not the SRK) is used, you must specify its handle index using `--tpm2-seal-key-handle=`.

The systemd-tpm2-setup.service(8) service writes the SRK to `/run/systemd/tpm2-srk-public-key.tpm2b_public` automatically during boot, in the correct format.

Alternatively, you may use **systemd-analyze srk** to retrieve the SRK from the TPM2 security chip explicitly. See systemd-analyze(1) for details. Example:

```
systemd-analyze srk > srk.tpm2b_public
```

Added in version 255.

`--tpm2-seal-key-handle=`*`HANDLE`*

Configures which parent key to use for sealing, using the TPM handle (index) of the key. This is used to "seal" (encrypt) a secret and must be used later to "unseal" (decrypt) the secret. Expects a hexadecimal 32bit integer, optionally prefixed with "`0x`". Allowable values are any handle index in the persistent ("`0x81000000`"-"`0x81ffffff`") or transient ("`0x80000000`"-"`0x80ffffff`") ranges. Since transient handles are lost after a TPM reset, and may be flushed during TPM context switching, they should not be used except for very specific use cases, e.g. testing.

The default is the Storage Root Key (SRK) handle index "`0x81000001`". A value of 0 will use the default. For the SRK handle, a new key will be created and stored in the TPM if one does not already exist; for any other handle, the key must already exist in the TPM at the specified handle index.

This should not be changed unless you know what you are doing.

Added in version 255.

`--tpm2-pcrs=`*`PCR[+PCR...]`*

Configures the TPM2 PCRs (Platform Configuration Registers) to bind to when enrollment is requested via `--tpm2-device=`. Takes a list of PCR entries, where each entry starts with a name or numeric index in the range 0…23, optionally followed by ":" and a hash algorithm name (specifying the PCR bank), optionally followed by "=" and a hash digest value. Multiple PCR entries are separated by "+". If not specified, the default is to use PCR 7 only. If an empty string is specified, binds the enrollment to no PCRs at all. See the table above for a list of available PCRs.

Example: `--tpm2-pcrs=boot-loader-code+platform-config+boot-loader-config` specifies that PCR registers 4, 1, and 5 should be used.

Example: `--tpm2-pcrs=7:sha256` specifies that PCR register 7 from the SHA256 bank should be used.

Example: `--tpm2-pcrs=4:sha1=3a3f780f11a4b49969fcaa80cd6e3957c33b2275` specifies that PCR register 4 from the SHA1 bank should be used, and a hash digest value of 3a3f780f11a4b49969fcaa80cd6e3957c33b2275 will be used instead of reading the current PCR value.

Added in version 248.

`--tpm2-with-pin=`*`BOOL`*

When enrolling a TPM2 device, controls whether to require the user to enter a PIN when unlocking the volume in addition to PCR binding, based on TPM2 policy authentication. Defaults to "`no`". Despite being called PIN, any character can be used, not just numbers.

Note that incorrect PIN entry when unlocking increments the TPM dictionary attack lockout mechanism, and may lock out users for a prolonged time, depending on its configuration. The lockout mechanism is a global property of the TPM, **systemd-cryptenroll** does not control or configure the lockout mechanism. You may use tpm2-tss tools to inspect or configure the dictionary attack lockout, with tpm2_getcap(1) and tpm2_dictionarylockout(1) commands, respectively.

Added in version 251.

`--tpm2-public-key=`*`PATH`*, `--tpm2-public-key-pcrs=`*`PCR[+PCR...]`*, `--tpm2-signature=`*`PATH`*

Configures a TPM2 signed PCR policy to bind encryption to. The `--tpm2-public-key=` option accepts a path to a PEM encoded RSA public key, to bind the encryption to. If this is not specified explicitly, but a file `tpm2-pcr-public-key.pem` exists in one of the directories `/etc/systemd/`, `/run/systemd/`, `/usr/lib/systemd/` (searched in this order), it is automatically used. The `--tpm2-public-key-pcrs=`

option takes a list of TPM2 PCR indexes to bind to (same syntax as `--tpm2-pcrs=` described above). If not specified defaults to 11 (i.e. this binds the policy to any unified kernel image for which a PCR signature can be provided).

Note the difference between `--tpm2-pcrs=` and `--tpm2-public-key-pcrs=`: the former binds decryption to the current, specific PCR values; the latter binds decryption to any set of PCR values for which a signature by the specified public key can be provided. The latter is hence more useful in scenarios where software updates shell be possible without losing access to all previously encrypted LUKS2 volumes. Like with `--tpm2-pcrs=`, names defined in the table above can also be used to specify the registers, for instance `--tpm2-public-key-pcrs=boot-loader-code+system-identity`.

The `--tpm2-signature=` option takes a path to a TPM2 PCR signature file as generated by the systemd-measure(1) tool. If this is not specified explicitly, a suitable signature file `tpm2-pcr-signature.json` is searched for in `/etc/systemd/`, `/run/systemd/`, `/usr/lib/systemd/` (in this order) and used. If a signature file is specified or found it is used to verify if the volume can be unlocked with it given the current PCR state, before the new slot is written to disk. This is intended as safety net to ensure that access to a volume is not lost if a public key is enrolled for which no valid signature for the current PCR state is available. If the supplied signature does not unlock the current PCR state and public key combination, no slot is enrolled and the operation will fail. If no signature file is specified or found no such safety verification is done.

Added in version 252.

`--tpm2-pcrlock=`*`PATH`*

Configures a TPM2 pcrlock policy to bind encryption to. Expects a path to a pcrlock policy file as generated by the systemd-pcrlock(8) tool. If a TPM2 device is enrolled and this option is not used but a file `pcrlock.json` is found in `/run/systemd/` or `/var/lib/systemd/` it is automatically used. Assign an empty string to turn this behaviour off.

Added in version 255.

Other Options

The following additional options are understood:

`--wipe-slot=`*`SLOT[,SLOT...]`*

Wipes one or more LUKS2 key slots. Takes a comma separated list of numeric slot indexes, or the special strings "`all`" (for wiping all key slots), "`empty`" (for wiping all key slots that are unlocked by an empty passphrase), "`password`" (for wiping all key slots that are unlocked by a traditional passphrase), "`recovery`" (for wiping all key slots that are unlocked by a recovery key), "`pkcs11`" (for wiping all key slots that are unlocked by a PKCS#11 token), "`fido2`" (for wiping all key slots that are unlocked by a FIDO2 token), "`tpm2`" (for wiping all key slots that are unlocked by a TPM2 chip), or any combination of these strings or numeric indexes, in which case all slots matching either are wiped. As safety precaution an operation that wipes all slots without exception (so that the volume cannot be unlocked at all anymore, unless the volume key is known) is refused.

This switch may be used alone, in which case only the requested wipe operation is executed. It may also be used in combination with any of the enrollment options listed above, in which case the enrollment is completed first, and only when successful the wipe operation executed — and the newly added slot is always excluded from the wiping. Combining enrollment and slot wiping may thus be used to update existing enrollments:

```
systemd-cryptenroll /dev/sda1 --wipe-slot=tpm2 --tpm2-device=auto
```

The above command will enroll the TPM2 chip, and then wipe all previously created TPM2 enrollments on the LUKS2 volume, leaving only the newly created one. Combining wiping and enrollment may also be used to replace enrollments of different types, for example for changing from a PKCS#11 enrollment to a FIDO2 one:

```
systemd-cryptenroll /dev/sda1 --wipe-slot=pkcs11 --fido2-device=auto
```

Or for replacing an enrolled empty password by TPM2:

```
systemd-cryptenroll /dev/sda1 --wipe-slot=empty --tpm2-device=auto
```

Added in version 248.

`--list-devices`

Show a list of candidate block devices this command may operate on. Specifically, this enumerates block devices currently present that contain a LUKS superblock, and shows their device node paths along with any of their symlinks. The devices must implement the `hmac-secret` extension to be useable.

Added in version 257.

`-h`, `--help`

Print a short help text and exit.

`--version`

Print a short version string and exit.

`--no-pager`

Do not pipe output into a pager.

Credentials

systemd-cryptenroll supports the service credentials logic as implemented by `ImportCredential=`/`LoadCredential=`/`SetCredential=` (see systemd.exec(5) for details). The following credentials are used when passed in:

`cryptenroll.passphrase`, `cryptenroll.new-passphrase`

May contain the passphrase to unlock the volume with/to newly enroll.

Added in version 256.

`cryptenroll.tpm2-pin`, `cryptenroll.new-tpm2-pin`

May contain the TPM2 PIN to unlock the volume with/to newly enroll.

Added in version 256.

`cryptenroll.fido2-pin`

If a FIDO2 token is enrolled this may contain the PIN of the token.

Added in version 256.

`cryptenroll.pkcs11-pin`

If a PKCS#11 token is enrolled this may contain the PIN of the token.

Added in version 256.

Exit status

On success, 0 is returned, a non-zero failure code otherwise.

Examples

crypttab(5) and systemd-measure(1) contain various examples employing **systemd-cryptenroll**.

See Also

systemd(1), systemd-cryptsetup@.service(8), crypttab(5), cryptsetup(8), systemd-measure(1)