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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'asymmetric-key.7' command

$ man asymmetric-key.7

ASYMMETRIC-KEY(7) Asymmetric Kernel Key Type ASYMMETRIC-KEY(7)

NAME

asymmetric - Kernel key type for holding asymmetric keys

OVERVIEW

A kernel key of asymmetric type acts as a handle to an asymmetric key as used for public-key cryptography. The key material itself may be held inside the kernel or it may be held in hardware with operations being offloaded. This prevents direct user access to the cryptographic material.

Keys may be any asymmetric type (RSA, ECDSA, ...) and may have both private and public components present or just the public component.

Asymmetric keys can be made use of by both the kernel and userspace. The kernel can make use of them for module signature verification and kexec image verification for example. Userspace is provided with a set of keyctl(KEYCTL_PKEY_*) calls for querying and using the key. These are wrapped by libkeyutils as functions named keyctl_pkey_*().

An asymmetric-type key can be loaded by the keyctl utility using a com? mand line like:

```
openssl x509 -in key.x509 -outform DER |
keyctl padd asymmetric foo @s
```

DESCRIPTION

The asymmetric-type key can be viewed as a container that comprises of a number of components:

Parsers

The asymmetric key parsers attempt to identify the content of the payload blob and extract useful data from it with which to instantiate the key. The parser is only used when adding, in? stantiating or updating a key and isn't thereafter associated with the key.

Available parsers include ones that can deal with DER-encoded X.509, DER-encoded PKCS#8 and DER-encoded TPM-wrapped blobs.

Public and private keys

These are the cryptographic components of the key pair. The public half should always be available, but the private half might not be. What operations are available can be queried, as can the size of the key. The key material may or may not actu? ally reside in the kernel.

Identifiers

In addition to the normal key description (which can be gener? ated by the parser), a number of supplementary identifiers may be available that can be searched for. These may be obtained, for example, by hashing the public key material or from the sub? jectKeyIdentifier in an X.509 certificate.

Identifier-based searches are selected by passing as the de? scription to keyctl_search() a string constructed of hex charac? ters prefixed with either "id:" or "ex:". The "id:" prefix in? dicates that a partial tail match is permissible whereas "ex:" requires an exact match on the full string. The hex characters indicate the data to match.

Subtype

This is the driver inside the kernel that accesses the key mate? rial and performs operations on it. It might be entirely soft? ware-based or it may offload the operations to a hardware key store, such as a TPM.

Note that expiry times from the payload are ignored as these patches may be used during boot before the system clock is set.

PARSERS

The asymmetric key parsers can handle keys in a number of forms:

X.509 DER-encoded X.509 certificates can be accepted. Two identifiers are constructed: one from from the certificate issuer and serial number and the other from the subjectKeyIdentifier, if present. If left blank, the key description will be filled in from the subject field plus either the subjectKeyIdentifier or the seri? alNumber. Only the public key is filled in and only the encrypt and verify operations are supported.

The signature on the X.509 certificate may be checked by the keyring it is being added to and it may also be rejected if the key is blacklisted.

PKCS#8 Unencrypted DER-encoded PKCS#8 key data containers can be ac? cepted. Currently no identifiers are constructed. The private key and the public key are loaded from the PKCS#8 blobs. En? crypted PKCS#8 is not currently supported.

TPM-Wrapped keys

DER-encoded TPM-wrapped TSS key blobs can be accepted. Cur? rently no identifiers are constructed. The public key is ex? tracted from the blob but the private key is expected to be res? ident in the TPM. Encryption and signature verification is done in software, but decryption and signing are offloaded to the TPM so as not to expose the private key.

This parser only supports TPM-1.2 wrappings and enc=pkcs1 encod? ing type. It also uses a hard-coded null SRK password; pass? word-protected SRKs are not yet supported.

USERSPACE API

In addition to the standard keyutils library functions, such as keyctl_update(), there are five calls specific to the asymmetric key type (though they are open to being used by other key types also):

keyctl_pkey_query()

keyctl_pkey_encrypt()

keyctl_pkey_decrypt()

keyctl_pkey_sign()

keyctl_pkey_verify()

The query function can be used to retrieve information about an asym? metric key, such as the key size, the amount of space required by buf? fers for the other operations and which operations are actually sup? ported.

The other operations form two pairs: encrypt/decrypt and create/verify signature. Not all of these operations will necessarily be available; typically, encrypt and verify only require the public key to be avail? able whereas decrypt and sign require the private key as well.

All of these operations take an information string parameter that sup? plies additional information such as encoding type/form and the pass? word(s) needed to unlock/unwrap the key. This takes the form of a comma-separated list of "key[=value]" pairs, the exact set of which de? pends on the subtype driver used by a particular key.

Available parameters include:

enc=<type>

The encoding type for use in an encrypted blob or a signature. An example might be "enc=pkcs1".

hash=<name>

The name of the hash algorithm that was used to digest the data to be signed. Note that this is only used to construct any en? coding that is used in a signature. The data to be signed or verified must have been parsed by the caller and the hash passed to keyctl_pkey_sign() or keyctl_pkey_verify() beforehand. An example might be "hash=sha256".

Note that not all parameters are used by all subtypes.

RESTRICTED KEYRINGS

An additional keyutils function, keyctl_restrict_keyring(), can be used to gate a keyring so that a new key can only be added to the affected keyring if (a) it's an asymmetric key, (b) it's validly signed by a key in some appropriate keyring and (c) it's not blacklisted.

keyctl_restrict_keyring(keyring, "asymmetric",

"key_or_keyring:<signing-key>[:chain]");

Where <signing-key> is the ID of a key or a ring of keys that act as the authority to permit a new key to be added to the keyring. The chain flag indicates that keys that have been added to the keyring may also be used to verify new keys. Authorising keys must themselves be asymmetric-type keys that can be used to do a signature verification on the key being added.

Note that there are various system keyrings visible to the root user that may permit additional keys to be added. These are typically gated by keys that already exist, preventing unauthorised keys from being used for such things as module verification.

BLACKLISTING

When the attempt is made to add a key to the kernel, a hash of the pub? lic key is checked against the blacklist. This is a system keyring named .blacklist and contains keys of type blacklist. If the blacklist contains a key whose description matches the hash of the new key, that new key will be rejected with error EKEYREJECTED.

The blacklist keyring may be loaded from multiple sources, including a list compiled into the kernel and the UEFI dbx variable. Further hashes may also be blacklisted by the administrator. Note that black? listing is not retroactive, so an asymmetric key that is already on the system cannot be blacklisted by adding a matching blacklist entry later.

VERSIONS

The asymmetric key type first appeared in v3.7 of the Linux kernel, the restriction function in v4.11 and the public key operations in v4.20.

SEE ALSO

keyctl(1), add_key(2), keyctl(3), keyctl_pkey_encrypt(3), keyctl_pkey_query(3), keyctl_pkey_sign(3), keyrings(7), keyutils(7)