

**NAME**

`add_key` – add a key to the kernel’s key management facility

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <keyutils.h>
```

```
key_serial_t add_key(const char *type, const char *description,
                    const void *payload, size_t plen,
                    key_serial_t keyring);
```

No glibc wrapper is provided for this system call; see NOTES.

**DESCRIPTION**

`add_key()` creates or updates a key of the given *type* and *description*, instantiates it with the *payload* of length *plen*, attaches it to the nominated *keyring*, and returns the key’s serial number.

The key may be rejected if the provided data is in the wrong format or it is invalid in some other way.

If the destination *keyring* already contains a key that matches the specified *type* and *description*, then, if the key type supports it, that key will be updated rather than a new key being created; if not, a new key (with a different ID) will be created and it will displace the link to the extant key from the keyring.

The destination *keyring* serial number may be that of a valid keyring for which the caller has *write* permission. Alternatively, it may be one of the following special keyring IDs:

**KEY\_SPEC\_THREAD\_KEYRING**

This specifies the caller’s thread-specific keyring (**thread-keyring(7)**).

**KEY\_SPEC\_PROCESS\_KEYRING**

This specifies the caller’s process-specific keyring (**process-keyring(7)**).

**KEY\_SPEC\_SESSION\_KEYRING**

This specifies the caller’s session-specific keyring (**session-keyring(7)**).

**KEY\_SPEC\_USER\_KEYRING**

This specifies the caller’s UID-specific keyring (**user-keyring(7)**).

**KEY\_SPEC\_USER\_SESSION\_KEYRING**

This specifies the caller’s UID-session keyring (**user-session-keyring(7)**).

**Key types**

The key *type* is a string that specifies the key’s type. Internally, the kernel defines a number of key types that are available in the core key management code. Among the types that are available for user-space use and can be specified as the *type* argument to `add_key()` are the following:

*"keyring"*

Keyrings are special key types that may contain links to sequences of other keys of any type. If this interface is used to create a keyring, then *payload* should be NULL and *plen* should be zero.

*"user"*

This is a general purpose key type whose payload may be read and updated by user-space applications. The key is kept entirely within kernel memory. The payload for keys of this type is a blob of arbitrary data of up to 32,767 bytes.

*"logon"* (since Linux 3.3)

This key type is essentially the same as *"user"*, but it does not permit the key to read. This is suitable for storing payloads that you do not want to be readable from user space.

This key type vets the *description* to ensure that it is qualified by a "service" prefix, by checking to ensure that the *description* contains a ':' that is preceded by other characters.

*"big\_key"* (since Linux 3.13)

This key type is similar to *"user"*, but may hold a payload of up to 1 MiB. If the key payload is large enough, then it may be stored encrypted in tmpfs (which can be swapped out) rather than kernel memory.

For further details on these key types, see **keyrings(7)**.

## RETURN VALUE

On success, **add\_key()** returns the serial number of the key it created or updated. On error, `-1` is returned and *errno* is set to indicate the cause of the error.

## ERRORS

### EACCES

The keyring wasn't available for modification by the user.

### EDQUOT

The key quota for this user would be exceeded by creating this key or linking it to the keyring.

### EFAULT

One or more of *type*, *description*, and *payload* points outside process's accessible address space.

### EINVAL

The size of the string (including the terminating null byte) specified in *type* or *description* exceeded the limit (32 bytes and 4096 bytes respectively).

### EINVAL

The payload data was invalid.

### EINVAL

*type* was "logon" and the *description* was not qualified with a prefix string of the form "service:".

### EKEYEXPIRED

The keyring has expired.

### EKEYREVOKED

The keyring has been revoked.

### ENOKEY

The keyring doesn't exist.

### ENOMEM

Insufficient memory to create a key.

### EPERM

The *type* started with a period ('.'). Key types that begin with a period are reserved to the implementation.

### EPERM

*type* was "keyring" and the *description* started with a period ('.'). Keyrings with descriptions (names) that begin with a period are reserved to the implementation.

## VERSIONS

This system call first appeared in Linux 2.6.10.

## CONFORMING TO

This system call is a nonstandard Linux extension.

## NOTES

No wrapper for this system call is provided in glibc. A wrapper is provided in the *libkeyutils* package. When employing the wrapper in that library, link with *-lkeyutils*.

## EXAMPLE

The program below creates a key with the type, description, and payload specified in its command-line arguments, and links that key into the session keyring. The following shell session demonstrates the use of the program:

```
$ ./a.out user mykey "Some payload"
Key ID is 64a4dca
$ grep '64a4dca' /proc/keys
064a4dca I--Q--- 1 perm 3f010000 1000 1000 user mykey: 12
```

**Program source**

```

#include <sys/types.h>
#include <keyutils.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int
main(int argc, char *argv[])
{
    key_serial_t key;

    if (argc != 4) {
        fprintf(stderr, "Usage: %s type description payload\n",
            argv[0]);
        exit(EXIT_FAILURE);
    }

    key = add_key(argv[1], argv[2], argv[3], strlen(argv[3]),
        KEY_SPEC_SESSION_KEYRING);
    if (key == -1) {
        perror("add_key");
        exit(EXIT_FAILURE);
    }

    printf("Key ID is %lx\n", (long) key);

    exit(EXIT_SUCCESS);
}

```

**SEE ALSO**

**keyctl(1), keyctl(2), request\_key(2), keyctl(3), keyrings(7), keyutils(7), persistent-keyring(7), process-keyring(7), session-keyring(7), thread-keyring(7), user-keyring(7), user-session-keyring(7)**

The kernel source files *Documentation/security/keys/core.rst* and *Documentation/keys/request-key.rst* (or, before Linux 4.13, in the files *Documentation/security/keys.txt* and *Documentation/security/keys-request-key.txt*).

**COLOPHON**

This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.