NAME

zfs-load-key - load, unload, or change encryption key of ZFS dataset

SYNOPSIS

zfs load-key [**-nr**] [**-L** *keylocation*] **-a**|*filesystem*
zfs unload-key [**-r**] **-a**|*filesystem*
zfs change-key [**-l**] [**-o** *keylocation=value*] [**-o** *keyformat=value*] [**-o** *pbkdf2iters=value*] *filesystem*
zfs change-key -i [**-l**] *filesystem*

DESCRIPTION

zfs load-key [**-nr**] [**-L** *keylocation*] **-a**|*filesystem*

Load the key for *filesystem*, allowing it and all children that inherit the **keylocation** property to be accessed. The key will be expected in the format specified by the **keyformat** and location specified by the **keylocation** property. Note that if the **keylocation** is set to **prompt** the terminal will interactively wait for the key to be entered. Loading a key will not automatically mount the dataset. If that functionality is desired, **zfs mount -l** will ask for the key and mount the dataset (see zfs-mount(8)). Once the key is loaded the **keystatus** property will become **available**.

-r Recursively loads the keys for the specified filesystem and all descendent encryption roots.

-a

Loads the keys for all encryption roots in all imported pools.

-n

Do a dry-run ("No-op") **load-key**. This will cause **zfs** to simply check that the provided key is correct. This command may be run even if the key is already loaded.

-L *keylocation*

Use *keylocation* instead of the **keylocation** property. This will not change the value of the property on the dataset. Note that if used with either **-r** or **-a**, *keylocation* may only be given as **prompt**.

zfs unload-key [**-r**] **-a**|*filesystem*

Unloads a key from ZFS, removing the ability to access the dataset and all of its children that inherit the **keylocation** property. This requires that the dataset is not currently open or mounted. Once the key is unloaded the **keystatus** property will become **unavailable**.

-r Recursively unloads the keys for the specified filesystem and all descendent encryption roots.

-a

Unloads the keys for all encryption roots in all imported pools.

zfs change-key [**-l**] [**-o** *keylocation=value*] [**-o** *keyformat=value*] [**-o** *pbkdf2iters=value*] *filesystem*

zfs change-key -i [**-l**] *filesystem*

Changes the user's key (e.g. a passphrase) used to access a dataset. This command requires that the existing key for the dataset is already loaded. This command may also be used to change the **keylocation**, **keyformat**, and **pbkdf2iters** properties as needed. If the dataset was not previously an encryption root it will become one. Alternatively, the **-i** flag may be provided to cause an encryption root to inherit the parent's key instead.

If the user's key is compromised, **zfs change-key** does not necessarily protect existing or newly-written data from attack. Newly-written data will continue to be encrypted with the same master key as the existing data. The master key is compromised if an attacker obtains a user key and the corresponding wrapped master key. Currently, **zfs change-key** does not overwrite the previous wrapped master key on disk, so it is accessible via forensic analysis for an indeterminate length of time.

In the event of a master key compromise, ideally the drives should be securely erased to remove all the old data (which is readable using the compromised master key), a new pool created, and the data copied back. This can be approximated in place by creating new datasets, copying the data (e.g. using **zfs send** | **zfs recv**), and then clearing the free space with **zpool trim --secure** if supported by your hardware, otherwise **zpool initialize**.

-l Ensures the key is loaded before attempting to change the key. This is effectively equivalent to running **zfs load-key** *filesystem*; **zfs change-key** *filesystem*

-o *property=value*
Allows the user to set encryption key properties (**keyformat**, **keylocation**, and **pbkdf2iters**) while changing the key. This is the only way to alter **keyformat** and **pbkdf2iters** after the dataset has been created.

-i Indicates that zfs should make *filesystem* inherit the key of its parent. Note that this command can only be run on an encryption root that has an encrypted parent.

Encryption

Enabling the **encryption** feature allows for the creation of encrypted filesystems and volumes. ZFS will encrypt file and volume data, file attributes, ACLs, permission bits, directory listings, FUID mappings, and **userused**/**groupused** data. ZFS will not encrypt metadata related to the pool structure, including dataset and snapshot names, dataset hierarchy, properties, file size, file holes, and deduplication tables (though the deduplicated data itself is encrypted).

Key rotation is managed by ZFS. Changing the user's key (e.g. a passphrase) does not require re-

encrypting the entire dataset. Datasets can be scrubbed, resilvered, renamed, and deleted without the encryption keys being loaded (see the **load-key** subcommand for more info on key loading).

Creating an encrypted dataset requires specifying the **encryption** and **keyformat** properties at creation time, along with an optional **keylocation** and **pbkdf2iters**. After entering an encryption key, the created dataset will become an encryption root. Any descendant datasets will inherit their encryption key from the encryption root by default, meaning that loading, unloading, or changing the key for the encryption root will implicitly do the same for all inheriting datasets. If this inheritance is not desired, simply supply a **keyformat** when creating the child dataset or use **zfs change-key** to break an existing relationship, creating a new encryption root on the child. Note that the child's **keyformat** may match that of the parent while still creating a new encryption root, and that changing the **encryption** property alone does not create a new encryption root; this would simply use a different cipher suite with the same key as its encryption root. The one exception is that clones will always use their origin's encryption key. As a result of this exception, some encryption-related properties (namely **keystatus**, **keyformat**, **keylocation**, and **pbkdf2iters**) do not inherit like other ZFS properties and instead use the value determined by their encryption root. Encryption root inheritance can be tracked via the read-only **encryptionroot** property.

Encryption changes the behavior of a few ZFS operations. Encryption is applied after compression so compression ratios are preserved. Normally checksums in ZFS are 256 bits long, but for encrypted data the checksum is 128 bits of the user-chosen checksum and 128 bits of MAC from the encryption suite, which provides additional protection against maliciously altered data. Deduplication is still possible with encryption enabled but for security, datasets will only deduplicate against themselves, their snapshots, and their clones.

There are a few limitations on encrypted datasets. Encrypted data cannot be embedded via the **embedded_data** feature. Encrypted datasets may not have **copies**=*3* since the implementation stores some encryption metadata where the third copy would normally be. Since compression is applied before encryption, datasets may be vulnerable to a CRIME-like attack if applications accessing the data allow for it. Deduplication with encryption will leak information about which blocks are equivalent in a dataset and will incur an extra CPU cost for each block written.

SEE ALSO

zfsprops(7), zfs-create(8), zfs-set(8)