datto

Encryption at Rest in ZFS

Tom Caputi
tcaputi@datto.com

Overview of Encryption Implementation

What is Encryption?

- **Want to prevent someone (an attacker) from accessing private data**
- **Permissions aren't good enough**
  - Root user can always access every file
  - Kernel bugs can result in privilege escalation
  - Disks can always be moved to another machine / OS and read
- **Solution: Encryption**
  - Data on disk should look pseudorandom (no detectable patterns)
  - User has a secret key that can be used to access the data
  - Mathematically, data is extremely hard to decrypt without key

Problems with Non-Native Encryption

- **File Level Encryption** (eg. ecryptfs)
  - Encryption before compression -> no compression
  - No dedup capabilities (within dataset)
  - Writes a metadata header, can disturb file alignment or waste space
- **Disk Level Encryption** (eg. dm-crypt)
  - Multiple copies are encrypted multiple times
  - Keys must always be loaded or pool is useless
    - No scrub, resilver, etc
    - No possibility of doing `zfs send` without keys loaded

- **Complex management**

How is this important to Datto?

- **Our primary backup solution for our partners**
  - A backup agent runs on our client's machines
  - Backups are sent to our device on the client's network
  - Backups are replicated to servers in the cloud (`zfs send`)

- **Advantages of Native Encryption**
  - Higher performance encryption, without losing compression
  - Much cleaner implementation than current stacked block devices
  - Ability to backup customer data without liability

What is Encrypted?

Encrypted

- File data and metadata
  - ACLs, names, permissions, attrs
- Directory listings
- All Zvol data
- FUID Mappings
- Master encryption keys
- All of the above in the L2ARC
- All of the above in the ZIL

Not Encrypted

- Dataset / snapshot names
- Dataset properties
- Pool layout
- ZFS Structure
- Dedup tables
- Everything in RAM

Caveats of Native ZFS Encryption

- Limited to `copies=2`
- Dedup tables are not encrypted
  - Dedup will leak data about equivalent data blocks
  - Dedup will only work within “clone families”
- Encryption + compression could allow for a CRIME attack
  - Not relevant to most applications
  - Can be prevented with `compression=off`

Creating an Encrypted Dataset

```
zfs create \
   -o encryption=on                \
   -o keylocation=prompt           \
   -o keyformat=passphrase         \
   -o pbkdf2iters=300000           \
   pool/encrypted_ds
```

ZFS Encryption Properties: Encryption

- **Encryption**
  - `off | on | aes-<128|192|256>-<ccm|gcm>`
  - Determines the cipher suite used to encrypt data
  - Currently AES-GCM and AES CCM are the only supported modes
  - Read-only after creation
  - `on` will default to `aes-256-ccm`
  - Children will inherit this property from parents
  - Will NOT adapt with newer revisions (unlike compression)

ZFS Encryption Properties: Keylocation

- **Keylocation**
  - `prompt | file://</absolute/path/to/keyfile>`
  - Determines where the key should be loaded from by default
  - `prompt` allows you to pipe keys into zfs for extensibility
  - Child datasets inherit the parent's keylocation
    - Inherited keylocation implies a common user key
    - ZFS will automatically allow you to mount inheriting datasets
    - Child datasets with a local key location will use a different user key

ZFS Encryption Properties: Keystatus

- **Keystatus**
  - `none | available | unavailable`
  - Tells whether a dataset's key is loaded
  - Changed by loading / unloading the key with the keystore commands
  - When the key is loaded filesystems and zvols function normally
  - When the key is not loaded
    - Zvols will not appear in /dev/
    - Filesystems will not be mountable

ZFS Encryption Properties: Keyformat

- **Keyformat**
  - `passphrase | hex | raw`
  - Determines the format of your key
  - `passphrase` may be between 8 and 64 characters
  - `hex` and `raw` keys must represent 32 bytes of random data

- **Pbkdf2iters**
  - `Integer value`
  - Allows tuning performance/security of passphrases
  - Only used when `keyformat=passphrase`
  - Default value is 350,000 Minimum value is 100,000

Creating an Encrypted Dataset

```
zfs create \
   -o encryption=<cipher suite>  \
   -o keylocation=<key location> \
   -o keyformat=<key format>     \
   [-o pbkdf2iters=<value>]      \
   <dataset name>
```

- Creates a new encryption root with its own user key
- Automatically loads the user key into ZFS

Loading and Unloading User Keys

`zfs unload-key <dataset>`

- `-a` and `-r` for recursive key unloading
- Fails if dataset is mounted or in any way open

`zfs load-key <dataset>`

- `-a` and `-r` for recursive key loading
- `-n` (no-op) for verifying a key is correct
- `-L <key location>` for loading the key from an alternate keylocation

Changing User Keys

```
zfs change-key [-li] \
   [-o keylocation=<key location>] \
   [-o keyformat=<key format>]     \
   [-o pbkdf2iters=<value>]        \
   <dataset name>
```

- Changes the user key of a dataset
- Will not re-encrypt all of the data
- Will cause the dataset to become an encryption root
- `-l` (load-key) ensures the key is loaded before changing it
- `-i` (inherit) will cause the dataset to inherit its parent's user key

Scrubs and Resilvering

- ZFS can still maintain integrity even without loaded keys
  - ZFS maintains checksums of the ciphertext
  - This includes:
    - Pool importing
    - Scrubs
    - Resilvers / Drive replacements
    - Self healing
    - Disk error reporting

Raw Sends

`zfs send -r`

- Enables sending encrypted data without loading the user key
- Data will be sent compressed and encrypted exactly as it is on disk
- Compatible with other send features such as `-D` and `-L`
- Allows backups to reside on an untrusted server
  - Better data protection for customers
  - Better liability prevention for data storage companies

ZFS Keystore API Overview

- **Primary API**
  - `zfs create, zfs clone`
    - `-o encryption=<cipher suite>`
    - `-o keylocation=<keylocation>`
    - `-o keyformat=<keyformat>`
    - `-o pbkdf2iters=<value>`
  - `zfs load-key, zfs unload-key`
  - `zfs change-key`
  - `zfs send -r`
- **Smaller changes**
  - `zfs mount, zpool import`

Current Status

- Fully implemented and feature complete
- Currently under review
- Pull requests are out for Linux, OSX, Illumos
  - Primary PR is on Linux

- **Special Thanks**
  - Jorgen Lundman for maintaining the ports to OSX and Illumos
  - Matt Ahrens and Brian Behlendorf for all the help answering my questions
  - George Wilson and Dan Kimmel for helping me through the ARC changes

datto

Questions?

Tom Caputi
tcaputi@datto.com
https://github.com/zfsonlinux/zfs/pull/5769