Encryption for filesystems with advanced features

Sweet Tea Dorminy

FACEBOOK Infrastructure

Agenda

Intro: fscrypt, motivation, extent-based fscrypt

Extent-based encryption: status, learnings, solutions

Future goals: more features from LUKS and bcachefs

Intro

Advanced filesystems?

- Not a judgement on quality, just a convenient alias for a particular set of features.
- Reflinks, subvolumes, snapshots, checksums
- Btrfs, XFS, Bcachefs*

What is fscrypt?

- Kernel library providing a standard encryption interface across filesystems using it.
- Used on Android
- Ext4, f2fs, ceph, ubifs as yet
- One master key per directory tree
- No mixing keys within one tree
- Can delete files without their key

- Only encrypts filenames and data (vs LUKS/dm-crypt, which encrypts everything)
- Crypto either with crypto api or blk-crypto
  - With blk-crypto, filesystem never sees encrypted data
- struct inode **embeds** struct fscrypt_inode_info, **storing on disk as** struct fscrypt_context:
  - Encryption is based on file + file offset
  - Key either master key+nonce applied to plaintext, or a derived key from master key + nonce

Difficulties for advanced filesystems

- **No mixing keys within one tree**
  - Breaks nested subvolumes with different master keys
- Crypto either with crypto api or blk-crypto
  - **With blk-crypto, filesystem never sees encrypted data**
    - Unsafe to store checksums of plaintext
- **struct inode embeds struct fscrypt_inode_info, storing on disk as struct fscrypt_context**
  - **Encryption is based on inode + file offset**
    - One piece of data can be reflinked into two inodes at different offsets. How to make both inodes decrypt it successfully? Awkward…

Motivation: btrfs

- Btrfs has long wanted to have encryption, but doesn’t want to give up checksumming or reflinking.
- By having per-subvolume encryption, individual user homedirs can have unique keys.

Extent-based encryption

- Still has a struct fscrypt_inode_info / struct fscrypt_context for inodes.
- **struct fscrypt_extent_context per extent**
  - **Encryption is based on extent + extent offset**
  - No issue reflinking an extent into two inodes anymore
  - Stores key, so in theory every extent can have a different key
- Takes more metadata space usually

Current state

History

- Design 1 in Oct ‘21 by Omar Sandoval
  - Per-extent context contained nonce only
  - Encryption using master key directly only
  - Patches Jun-Oct ‘22
  - Risks of master key reuse for too much data
  - Crypto api only
  - Checksum encrypted data
- Design 2 in Nov ‘22
  - Per-extent context reusing ‘normal’ per-inode context struct
  - Patches Jan-Aug ‘23
  - Insufficiently elegant
  - Blk-crypto only
  - Checksummed unencrypted data

Current state

- Design 3 in Sep ‘23
  - Per-extent context with nonce and key (must match inode key for now)
  - Encryption restricted to derived key from inode context + extent nonce
  - V2 in flight by Josef Bacik
  - Doesn’t support nested subvols with different keys or full range of key options, but enough information is in the context to do so
  - Checksum on encrypted data via callback in blk-crypto-fallback
  - Blk-crypto-fallback only
  - **Please review:**
    - https://lore.kernel.org/linux-fscrypt/cover.1696970227.git.josef@toxicpanda.com/T/#t

Addresses previous difficulties

- **Still doesn’t allow changing keys within one tree**
  - Nested subvolumes still don’t work, but enough info is stored to allow changing key between inodes.
- Extent-based only with blk-crypto
  - **Adds a callback to blk-crypto to allow checksumming encrypted data**
- **Encryption is based on inode + extent + extent offset**
  - **Addresses reflinking between inodes with the same key, can be extended to allow reflinking between inodes with different keys**

Future goals

Bcachefs has different features

- Doesn’t use fscrypt
- Only one encryption key per filesystem
- Everything is encrypted: no access to anything, even for deletion, when the key isn’t loaded
- Authenticated encryption instead of encryption + checksums of encrypted data
- Less options for encryption algorithm

LUKS (dm-crypt + dm-integrity) has different features

- Only one encryption key per filesystem
- Everything is encrypted: no access to anything, even for deletion, when the key isn’t loaded
- Authenticated encryption instead of encryption + checksums of encrypted data
- Encryption key changes
  - Useful for repudiation or changing to a newer encryption algorithm
- Encrypts everything

Key change motivation

- There’s a Fedora proposal to use btrfs encryption one day
  - initial unencrypted or encrypted image installed on disk
  - Company or user sets new key on /, installs own packages, sets up homedir template
  - User sets new key for homedir
- Meta once and may again want to install an unencrypted package in subvolume, run in container with per-subvolume key for anything written by package.

Key changes: how?

- Where/how to implement?
  - btrfs has per-subvol key for new extents, online or offline update of that key?
  - new extents inherit inode context, kernelspace recursively updates inode contexts in directory tree?
  - new extents inherit inode context, userspace recursively calls kernelspace update of one inode context?
  - Interfaces are hard.

Authenticated Encryption

- Detects corruption in a cryptographically clever way, get EIO instead of corrupt data
- Store a nonce and a ‘authentication tag’ (like a checksum)
- btrfs uniquely positioned since it already has per-block metadata (storing a checksum).
- Currently used by dm-crypt+dm-integrity, but not in blk-crypto at present

Authenticated Encryption

- Complicates btrfs scrub/relocate, NOCOW, if auth tags are needed in normal IO path
- Plan to extend fscrypt to have a blk-integrity tag, and have blk-crypto fill that in with the authentication tag as needed

Integrate into more filesystems?

- What keeps your filesystem from using fscrypt?

FACEBOOK Infrastructure