Data integrity protection with cryptsetup tools

What is the Linux dm-integrity module and why we extended dm-crypt to use authenticated encryption.

Milan Brož
gmazyland@gmail.com

FOSDEM, Brussels
February 4, 2018
Agenda

- Data integrity protection and disk encryption?
- dm-integrity and dm-crypt Linux kernel modules
- dm-integrity standalone mode
- dm-crypt authenticated encryption
- LUKS2
Full Disk Encryption (FDE)

- Disk sector level
  - Sectors accessed independently
  - 4k sector size today
- Data-at-rest protection
- Confidentiality
- Length-preserving encryption
  - plaintext size = ciphertext size
- No data integrity protection
FDE images with Tux

... not only kind of glitch-art :-)
FDE encryption example
AES-XTS, IV is sector offset

plaintext

ciphertext
Wrongly used modes, IVs, nonces, ciphertext patterns

ECB mode

XTS, constant IV
Length-preserving encryption
no integrity, garbage-in, garbage-out
FDE threat model?

- Stolen device, disk in repair, ...
  - Length-preserving, confidentiality only
  - Data never used again

- Our model: Returned device
  - Silent data corruption
  - Implanted data without owner knowledge

- Could this happen?
  - Lost disk returns to owner
  - Devices traveling separately...
Another FDE trade-offs

- Whole sector not pseudo-randomly changed on every write.
  - Granularity of ciphertext change
  - Same plaintext = same ciphertext (in the same sector)
  - Could we have randomized IVs?

- Replay attacks
  - Revert to old valid content
  - Need trusted store for root hash (Merkle tree)
Encryption block granularity
(each following block is inverted here)

AES block, 16B

4k disk sector
XTS Encryption block trade-off
Every 64 byte re-written, ciphertext diff.

AES-XTS, IV is sec#

The same byte was written :-)
What is missing?

- Confidentiality + data integrity protection
  => authenticated encryption (AEAD)

- Ciphertext change granularity
  => randomized IV (or wide encryption modes)

- Pseudo-random change on every write
  => randomized IV

- Additional metadata per-sector
FDE with data integrity protection

- FreeBSD GELI – different approach

- Our requirements
  - No special HW
  - Commercial of-the-shelf SSDs
  - Configurable per-sector metadata
  - Use native sector size
  - Reliable recovery on power fail
  - Algorithm agnostic
  - Free code & algorithms, no patents
Separation of storage and crypto

- **dm-integrity**
  - Emulates per-sector metadata
  - Optionally standalone mode (CRC32)
- **dm-crypt**
  - Authenticated encryption
  - Randomized IV
  - Tags and IVs stored in per-sector metadata
- **cryptsetup**
  - LUKS2 on-disk format
  - User friendly activation
dm-integrity on-disk layout

<table>
<thead>
<tr>
<th>Config and journal</th>
<th>Data and metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>Journal</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Data</td>
<td>...</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

- **Superblock (SB)** – persistent parameters

- **Journal area**
  - Can be deactivated (write performance penalty)

- **Metadata per 4k sector (packed)**
  - 32bits metadata (CRC32) – 0.1% of storage
  - 256bits metadata (SHA256) – 0.78% of storage
dm-integrity standalone mode

- Non-cryptographic data check-sums
  - Detects silent data corruption
  - CRC32 or hash

- Per-sector check-sum
  - Reads (validate) / Writes (update)

- No encryption of data

- Integritysetup tool
Integritysetup example

# integritysetup format /dev/sdb [ -I crc32c ]
Formatted with tag size 4, internal integrity crc32c.
Wiping device to initialize integrity checksum.

# integritysetup open /dev/sdb test [ -I crc32c ]

# integritysetup status test
  type:    INTEGRITY
  tag size: 4
  integrity: crc32c
  device:  /dev/sdb
  sector size:  512 bytes
  interleave sectors: 32768
  size:    2064392 sectors
  mode:    read/write
  journal size: 8380416 bytes
  journal watermark: 50%
  journal commit time: 10000 ms

# mkfs -t xfs /dev/mapper/test
....
dm-crypt authenticated encryption

- Authenticated request

| AAD         | DATA             | AUTH
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>authenticated</td>
<td>authenticated + encrypted</td>
<td>TAG</td>
</tr>
<tr>
<td>sector</td>
<td>data in/out</td>
<td>tag</td>
</tr>
<tr>
<td>IV</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Position must be authenticated
  - Misplaced sector
- Random IV (nonce)
  - On every write from RNG
  - Collision probability negligible (!)
- No protection to replay attacks
Authenticated algorithms

- No perfect algorithm in kernel for FDE!

- Length-preserving modes + HMAC (too slow)

- Authenticated modes
  - AES-GCM (96-bit nonce – collision is fatal)
  - ChaCha20-Poly1305 (RFC7539, 96-bit nonce)

- Future: CAESAR (crypto competition finalists)
  - AEGIS performs well

- Reason it remains experimental feature.
LUKS2 with integrity protection

```bash
# cryptsetup luksFormat --type luks2 /dev/sdb $PARAMS
$PARAMS AES-XTS+HMAC: --cipher aes-xts-plain64 --integrity hmac-sha256
$PARAMS ChaCha20-poly1305: --cipher chacha20-random --integrity poly1305

# cryptsetup open /dev/sdb test
# lsblk /dev/sdb
NAME        MAJ:MIN SIZE
sdb           8:16    1G
└─ test_dif  253:0   952M
  └─ test    253:1   952M

# cryptsetup status test
  type:    LUKS2
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  integrity: hmac_sha256
  integrity keysise: 256 bits
  device:  /dev/sdb
  sector size:  512
  size:    1949704 sectors
  mode:    read/write

  type:    LUKS2
  cipher:  chacha20_random
  keysize: 256 bits
  key location: keyring
  integrity: poly1305
  device:  /dev/sdb
  sector size:  512
  size:    1965064 sectors
  mode:    read/write

# cryptsetup close test
```
Performance (example: fio simulated)

- SSD, 30% writes / 70% reads (very inefficient case)
Summary

- Try it
  - cryptsetup 2.0.x, Linux kernel 4.12+

- We need new AEAD algorithms
- Integrity protection on higher layer better?

- dm-integrity in future?
  - Replaced by persistent memory
  - Variable sector with inline metadata.
LUKS2 (in cryptsetup2)

- LUKS is a key management
- LUKS2 is on-disk format for LUKS extensions
- Metadata replicated (not keyslots)
- JSON metadata
- Argon2 key derivation function
- Kernel keyring

- Cryptsetup does not handle HW tokens directly.
  => token concept (metadata + external program)

- LUKS1 supported forever :-(
LUKS2 & tokens

• Token – metadata object in header

  => How to get a passphrase for a keyslot.

1) Keyring token (internal)

• External app for HW (TPM, Smartcard, ...)

• Passphrase in kernel keyring

• Cryptsetup activation is automatic

2) External token types

• LUKS2 stores metadata

• External app uses libcryptsetup to activation

• Tokens ignored by cryptsetup
Thanks for your attention.

Q & A?

or use dm-crypt mailing list later

Milan Brož
gmazyland@gmail.com

FOSDEM, Brussels
February 4, 2018