https://wiki.archlinux.org/index.php/Dm-crypt/Encrypting_an_entire_system|dm-crypt/Encrypting an entire system

LUKS on LVM vs LVM on LUKS

LUKS on LVM Benefit:

1. Every logical volume is encrypted with separate password
2. good for multiuser environment
3. root system can be on unencrypted partition (no password to boot). The same can be achieved with LVM on LUKS on separate partition.
4. Volumes can span on multiple drives
5. LVM cache is caching encrypted data (no unecnrypted data leak to cache device).
    1. one common SSD cache device can be used if you have encrypted (data) and unecrypted (system) partitions on LVM

LVM on LUKS (preffered)

Block device is encrypted and on top of block device LVM is configured. Benefit:

1. one unlock of block device give access to all LVM volume created on it.
2. it is easier to change volumes sizes without touching encryption layer
3. LVM cache is caching decrypted data
    1. workaround: encrypt also cache device, but for mixed setup (unencrypted and crypted partition) it is need to divide cache device into 2 volumes to serve unencrypted cache for system (no need to provide unlock password).

Performance

IT depends on HW acceleration

```
cryptsetup benchmark
```

Best choice for AMD A4-5300 APU:

```
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1       448876 iterations per second
PBKDF2-sha256     352344 iterations per second
PBKDF2-sha512     362077 iterations per second
PBKDF2-ripemd160  500274 iterations per second
#  Algorithm | Key |  Encryption |  Decryption
     aes-cbc   128b   429.0 MiB/s  1275.9 MiB/s
     aes-cbc   256b   333.0 MiB/s   770.0 MiB/s
     aes-xts   256b   903.8 MiB/s  1023.9 MiB/s
     aes-xts   512b   902.7 MiB/s   928.5 MiB/s
```

Advices

Cipher

- AES well known, accelerated by common HW
- Twofish (faster SW implementation comparing to AES)

Chaining mode

- CBC. Every block will be XOR’ed with the encrypted previous block. This effectively means that every block depends on the output of the previous block. This mode is vulnerable to watermark attack, where attacker can inject own data to crypted block chain (for filesystem, access to block device is needed)
- EBC (Electronic Codebook), each block is encrypted with the same key
- XTS. Is counter-oriented chaining mode. It's an evolution of XEX (actually: “XEX-based tweaked-codebook mode with ciphertext stealing”), while XEX (“xor-encrypt-xor”) is a non-trivial counter-based chaining mode; neither of which I can claim to completely understand. XTS is already very widely supported and looks promising, but may have issues. The primary important details are these: No fancy IVs are necessary (plain or plain64 is fine), and half of your key is used by XTS, meaning your original key must be twice as long (hence 512-bit instead of 256-bit).

IV (Initalisation Vector) calculation

- plain
- plain64

Is an IV generation mechanism that simply passes the 64-bit sector index directly to the chaining algorithm as the IV. plain truncates that to 32-bit. Certain chaining modes such as XTS don't need the IV to be unpredictable, while modes like CBC would be vulnerable to fingerprinting/watermarking attacks if used with plain IVs.

- ESSIV

(“Encrypted salt-sector initialization vector”) allows the system to create IVs based on a hash including the sector number and encryption key. This allows you to jump straight to to the sector you want without resorting to predictable IVs, and therefore protects you from watermarking attacks.

LUKS' key derivation method

- SHA256

plain vs plain64

- Do not use plain for disc greater than 2TB (use plain64 instead)

XTS

- doesn't require ESSIV (aes-xts-plain)
- half of your key is used by XTS, meaning your original key must be twice as long (hence 512-bit instead of 256-bit).

CBC

- should be protected with ESSIV (aes-cbc-essiv)

If password are used instead of keyfile, to prevent brute force attack:

- choose very long password to prevent dictionary attacks.
- use big hash like SHA512 (–hash=sha512)
- increase number of iterations (default it is set to 1000 ms)

Fill with random data

```
badblocks -c 10240 -s -w -t random -v /dev/sda5
```

or (faster, only writes). Block size for dd has to be big, to avoid re-reading data from encrypted block.

```
cryptsetup open --type plain /dev/sda5 tempcontainer
dd if=/dev/zero of=/dev/mapper/tempcontainer bs=64M
cryptsetup luksClose tempcontainer
```

Setup /dev/sda5 as LUKS device:

```
cryptsetup luksFormat -y -v /dev/sda5
```

will create by default **aes-xts-plain64** 256bits.

Another examples:

```
cryptsetup luksFormat --cipher aes-cbc-plain --key-size 256 /dev/sda5
cryptsetup luksFormat --cipher aes-cbc-plain --key-size 256 --hash sha1 -i
2000 --use-random /dev/sda5
cryptsetup luksFormat --cipher aes-cbc-essiv:sha256 --key-size 256 --verify-
passphrase -v /dev/sda5
cryptsetup luksFormat --cipher aes-xts-plain --key-size 256 --verify-
passphrase -v /dev/sda5
cryptsetup luksFormat --cipher aes-xts-plain --key-size 512 --verify-
passphrase -v /dev/sda5
```

```
cryptsetup --verify-passphrase -v --cipher aes-cbc-plain64 --key-size 128 --
hash sha512 --iter-time 3000 --use-random luksFormat /dev/sda5
```

```
cryptsetup luksFormat --cipher aes-xts-plain --verify-passphrase -v  --key-
size 512  --hash sha512 --iter-time 3000 --use-random /dev/sdb6
```

Open LUKS device

```
cryptsetup luksOpen /dev/sda5 sda5
```

Examing status of LUKS

```
cryptsetup status sda5
cryptsetup luksDump /dev/sda5
```

Closing LUKS device

```
cryptsetup luksClose sda5
```

References

security.stackexchange.com/questions/40208/recommended-options-for-luks-cryptsetup

https://kiza.eu/journal/entry/697