Having Something To Hide

Trusted Key Storage in Linux

Kernel Devroom @ FOSDEM 2023

Ahmad Fatoum – a.fatoum@pengutronix.de

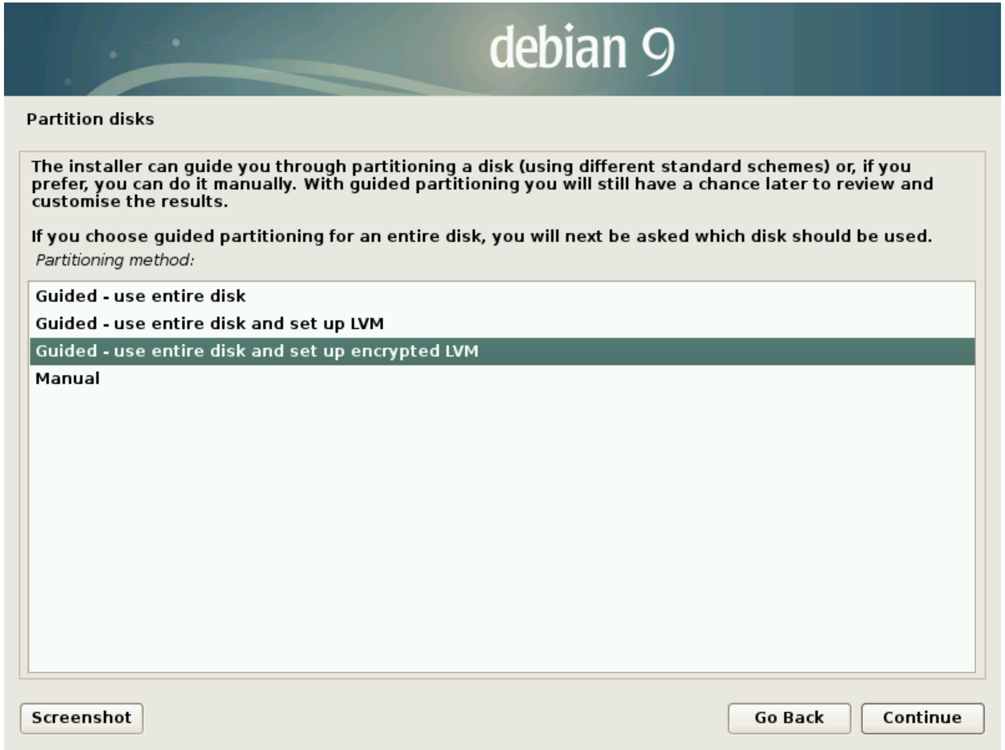
About Me

- Ahmad Fatoum
- Pengutronix e.K.
- a3f
- a.fatoum@pengutronix.de

- Kernel and Bootloader Porting
- Driver and Graphics Development
- System Integration
- Embedded Linux Consulting

Data encryption at rest

- Only decrypt partition at runtime
- Distro installers offer it with LUKS out of the box
- How does it work?



[https://xo.tc/setting-up-full-disk-encryption-on-debian-9-stretch.html]]

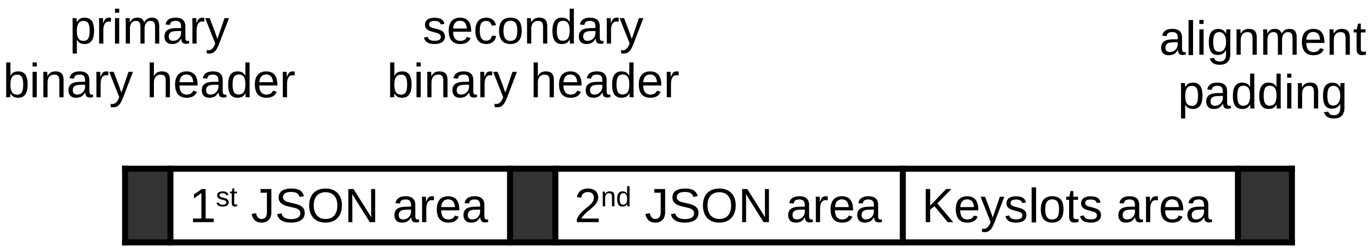
dm-crypt

- Device Mapper maps physical block devices onto virtual block devices
- **dm-crypt** target transparently encrypts virtual block device content to physical device

```
TABLE="                                      \
   0 $NBLOCKS crypt aes-cbc-essiv:sha256   \
   :32:logon:key 0 $DEV 0 1 allow_discards \
"
keyctl add logon key 01234567890123456789012345678912 @s
echo "$TABLE" | dmsetup create mydev
echo "$TABLE" | dmsetup load mydev
```

LUKS

- LUKS is a disk encryption specification for block devices
- dm-crypt volume key encrypted with one or more passphrases
- Encrypted keys persisted to LUKS keyslots area
- cryptsetup(1) is the usual implementation on Linux



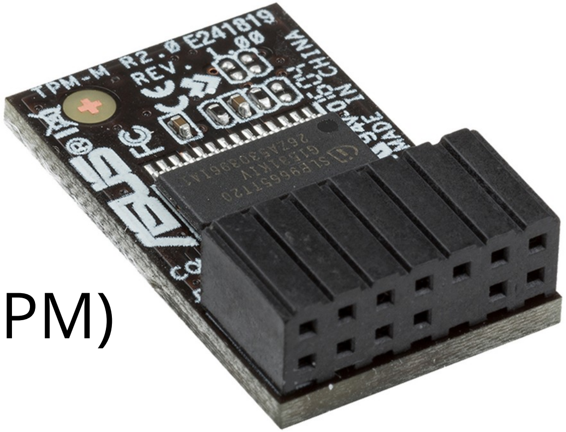
LUKS2 header on-disk structure

But where does the passphrase come from?

- User Input
  - User enters passphrase in initrd or attaches disk with keyfile
  - User inserts FIDO security key
  - User connects PKCS#11-compatible smart card
- What about unattended boots?
  - Trusted Storage needed to hold key and provide it to OS

Trusted Platform Modules (TPMs)

- TPM 1.2 standardized as ISO/IEC 11889
- TPM 2.0 mandated by Windows 11
- Available as discrete chips or as firmware (fTPM)
- Has random number generator built-in
- Holds unique never-disclosed key
    - Encrypts and decrypts data using this key
    - Decryption can be made conditional on integrity measurement (PCR sealing)



[https://www.amazon.de/-/en/Asus-TPM-M-R2-0-14-1-Module/dp/B01DQQLH74]

Utilizing TPMs from userspace

- Kernel provides `/dev/tpm`, `/dev/tpmrm` for direct and resource-managed access respectively
- Libraries exist: tpm2-tools by Intel and ibm-tss
- `systemd-cryptsetup` has native support for enrolling LUKS keys in TPMs: encrypted passphrase stored to LUKS2 JSON token area
- Keyphrase and dm-crypt key available to privileged userspace then stuffs dm-crypt key into kernel keyring
- Why not decrypt TPM-secured key directly into kernel keyring?

Linux Trusted and Encrypted Keys

- Trusted Keys have a hardware root of trust used to both generate and seal/unseal the keys
- Userspace sees, stores, and loads them only in encrypted form
- Encrypted Keys can be sealed with any key type
- Trusted Keys first added in 2010, originally TPM-specific

Trusted Keys + dm-crypt example

```
NBLOCKS=4096
TABLE="0 $NBLOCKS crypt aes-cbc-essiv:sha256 :32:trusted:kmk 0 /dev/loop0 0 1 allow_discards"
```

```
TKEY=$(keyctl add trusted kmk "new 32" @s)
keyctl pipe "$TKEY" >kmk.blob

fallocate -l $((NBLOCKS * 512)) loop.img
losetup -P /dev/loop0 loop.img

echo "$TABLE" | dmsetup create mydev
echo "$TABLE" | dmsetup load mydev
dd if=/dev/zero of=/dev/mapper/mydev || true
echo "It works!" 1<> /dev/mapper/mydev

cryptsetup close mydev
reboot
```

```
losetup -P /dev/loop0 loop.img

keyctl add trusted kmk \
    "load $(cat kmk.blob)" @s

echo "$TABLE" | dmsetup create mydev
echo "$TABLE" | dmsetup load mydev

# should print that It works!
hexdump -C /dev/mapper/mydev
```

Beyond TPMs

- Not everyone agrees it has advantages over doing it in userspace
  - But that's just because userspace TPM handling has enjoyed a *lot* of work
- Trusted Keys can be the interface of not just TPMs:
  - Off-Chip Secure Enclaves
  - On-Chip Trusted Execution Environments (TEE)
  - Crypto units inside your everyday SoCs
- Work started in 2019 to generalize Trusted Keys and add TEE support

Trusted Execution Environment

- GlobalPlatform API standard
- Hardware isolated environment hosts a number of trusted applications (TAs) making use of the API.
- TAs can implement fTPM, but all goes really:
  - Just RNG
  - Key sealing/unsealing with a hardware unique key
  - Clock, reset, power domain support, so Linux can't interfere with secure peripherals
  - `grep -r tee_client_driver /usr/src/linux`

CAAM

- NXP’s (née Freescale) Crypto Accelerator and Authentication Module
  - Available on the newer i.MX and QoriQ SoCs
- Linux already used it for RNG and Crypto Acceleration
- Direct Memory Access controlled via shared job rings
  - Shareable between Normal World (Linux) and Secure World (TEE in ARM TrustZone)
- Has access to a unique One-Time Programmable Master Key fused by NXP if High Assurance Boot is active
  - Red blob generation: Seal/Unseal user-supplied key material using the OTPMK
  - Black blob generation: Crypto done inside CAAM and key never disclosed

CAAM blobbing for Linux

- Common use case for red key blobbing: Certificate storage
- We had been carrying patches for many years across different customer kernels
- 2015: Proof of Concept sent to linux-crypto adding sysfs interface
- 2018: NXP suggests new „Secure“ key type specially for CAAM red blobbing
- 2019: NXP suggests new „trusted_tk“ key type specially for CAAM black blobbing
- 06/2019: RFC Trusted Key Framework generalization and TEE support
- 02/2021: v9 of TEE support accepted. Available since v5.13
- 07/2021: v1 of CAAM Trusted Keys Support
- 05/2022: v10 of CAAM support accepted. Available since v5.19

Upstreaming CAAM Trusted Key support

- TEE and TPM don't utilize the kernel entropy pool
  - CAAM driver could do likewise, but we have a perfectly fine CAAM RNG driver already
  - Some possible trust sources may not even have a random number generator (Example: i.MX6 UltraLiteLite DCP)

    → CAAM backend uses kernel entropy pool. New `trusted.rng=kernel` option enables this for other backends as well
- Hardware feature bits are broken on some variants
  - CAAMs exists that report BLOB support, but lack AES.. :-)

In-field migration without re-encryption

- Mainline „Trusted Keys“ CAAM blobs interchangable with vendor kernel „Secure Keys“
  - Thanks to upstreaming feedback
  - Makes life easier for users switching from vendor kernels
  - At the cost of making our own sysfs interface incompatible
- Use dm-crypt directly and exclude LUKS area
- One-time import step needed (non-upstream patch)

Trusted Keys: What more is there to do?

- Encrypted Key support (/key_type_encrypted/):
  - dm-crypt
  - eCryptFS
  - EVM
  - NVDIMM
- Direct Trusted Key support (/key_type_trusted/):
  - dm-crypt
  - Encrypted Keys
- Future candidates
  - fscrypt (keysetup v1 attempt, keysetup v2 attempt)
  - UBIFS authentication (First attempt here)
- LUKS Support would be awesome (Discussion)

Thanks!

Questions?