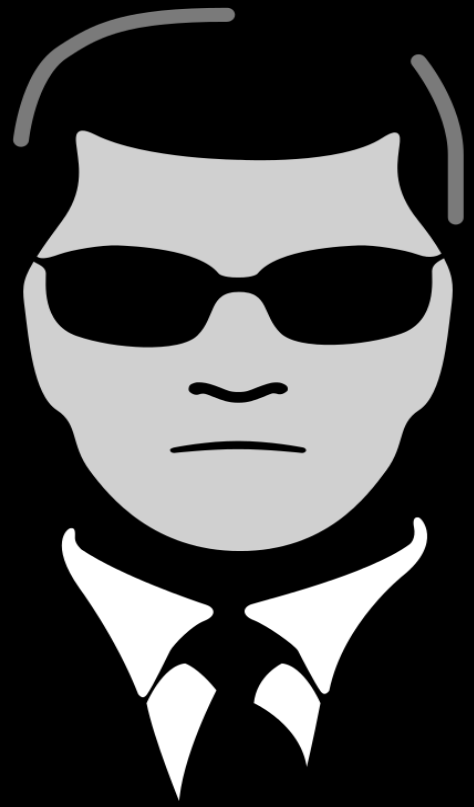
arm



Protecting your system from the scum of the universe

Embedded Linux Conference Europe

Gilad Ben-Yossef
gilad@benyossef.com
Twitter: @giladby

About me

My name is Gilad Ben-Yossef.

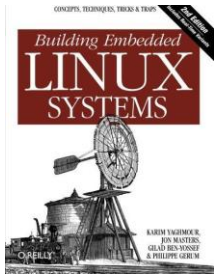
I work on applied cryptography and security of the upstream Linux kernel in general and maintain the arm® TrustZone® CryptoCell® Linux device driver.

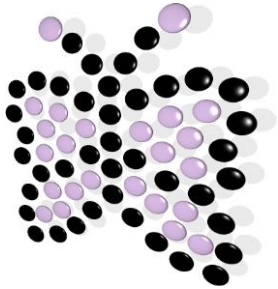
I have been working in various forms with and on the Linux kernel and other Open Source projects for close to twenty years.

I have co-authored “Building Embedded Linux Systems” 2nd edition from O’Reilly.

I’m a co-founder of HaMakor, an Israeli NPO for free and open source software and of August Penguin, the longest running Israeli FOSS conference.



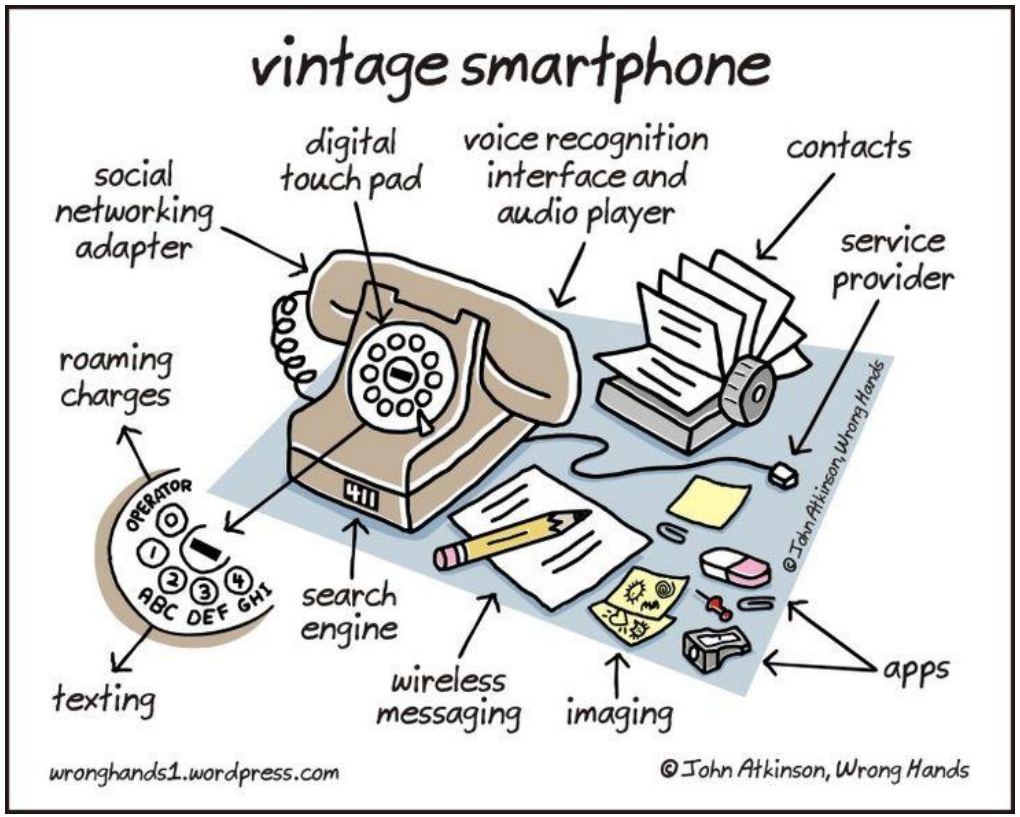
arm

BASED ON A TRUE STORY_

arm

Problem Definition

We use smart devices for everything



…and therefore need to trust them.

arm

What does “Trust” mean?

We want secure devices:

- Keep our secrets
- Serve us, not malicious strangers

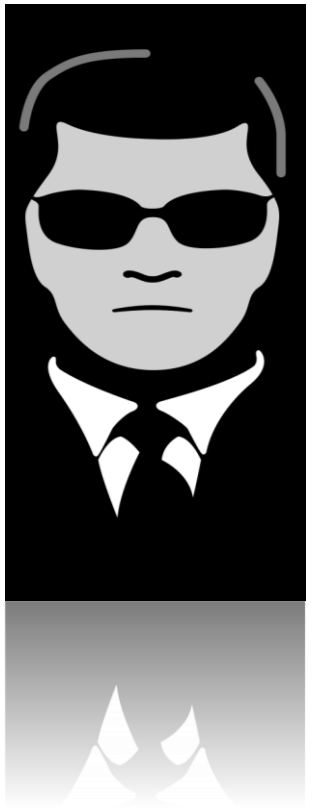
We also want:

- Connected devices
- Frictionless user experience
- Open ended devices

arm

A “Trusted” way of failing

- What do we want?
  - If someone got hold of our device, we don’t them to have access to (all) our secrets
  - If someone broke in, we don’t want them to be able to leverage this to gain access to additional resources we have access to.
  - If unauthorized changes occurred, we want to know about it.
  - If someone broke in, we don’t want them to take hold.
- The threat model of dealing with a malicious entity getting a temporary hold of our device:
  - Evil government agent getting hold of our device during customs inspection
  - Malware being run on our device



arm

How to build such a system?

Secure Boot

File System Integrity Verification (DM-Verity)

Full Disk Encryption (DM-Crypt)

File Based Encryption (Fscrypt)

Trusted Execution Environment

Trusted Platform Module

Integrity Measurement Architecture

How it all fits together

arm

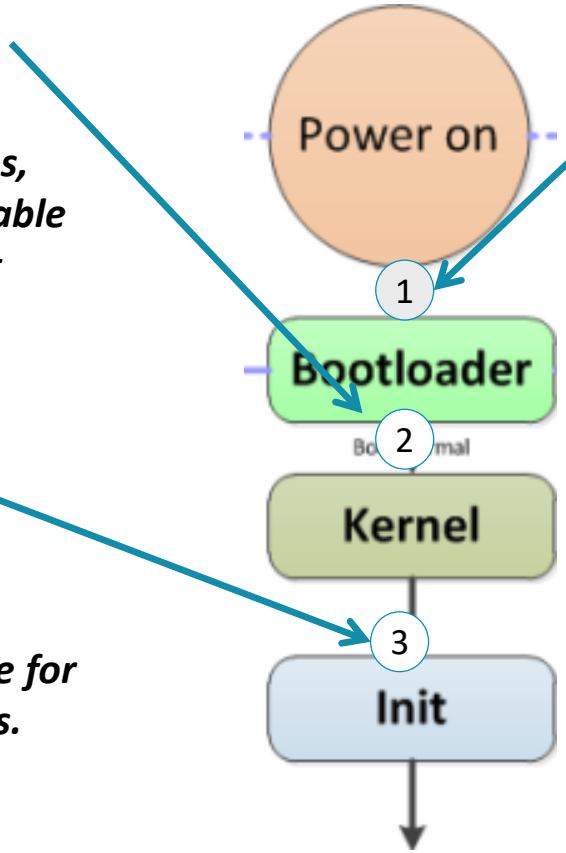
Secure Boot* Sequence

2 Boot loader verifies kernel image and boot file system

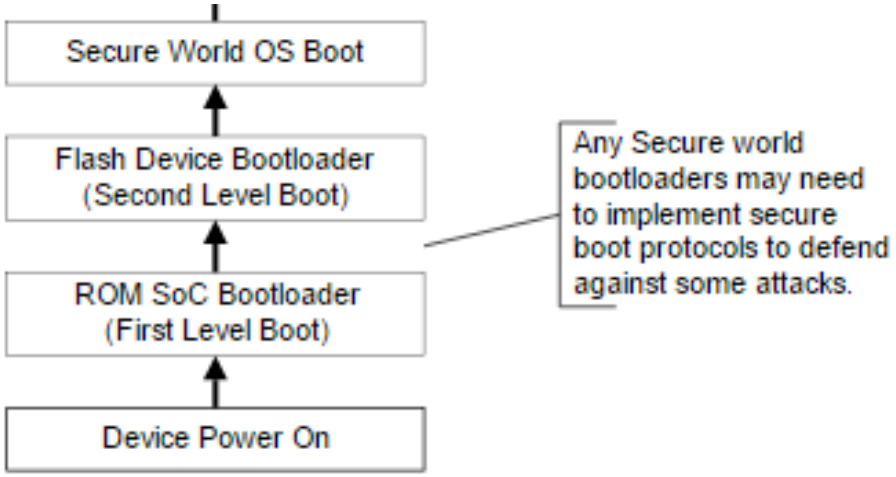
"For verifying boot and recovery partitions, the bootloader has a fixed OEM key available to it. It always attempts to verify the boot partition using the OEM key."

3 OS verifies the system and additional partitions

"Once execution has moved to the boot partition, the software there is responsible for setting up verification of further partitions. Due to its large size, the system partition typically cannot be verified similarly to previous parts but is verified as it's being accessed instead using the dm-verity kernel driver or a similar solution."



1 Firmware verifies the boot loader



"Bootloader integrity is always verified using a hardware root of trust."

* Android style

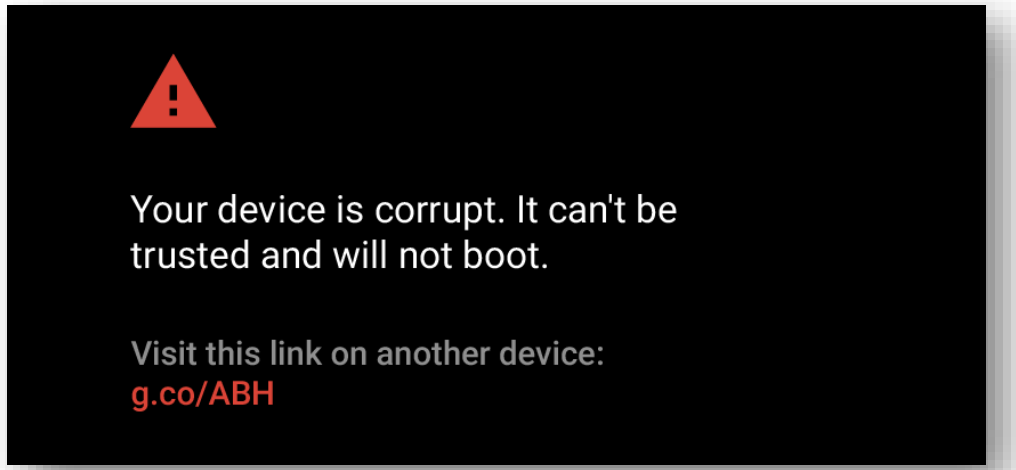
arm

Verified Boot (AKA DM-Verity)

Linux Device-Mapper's "verity" target provides transparent integrity checking of read only block devices.

DM-verity helps prevent persistent rootkits that can hold onto root privileges and compromise devices.

The DM-verity feature lets you look at a block device, the underlying storage layer of the file system, and determine if it matches its expected configuration.

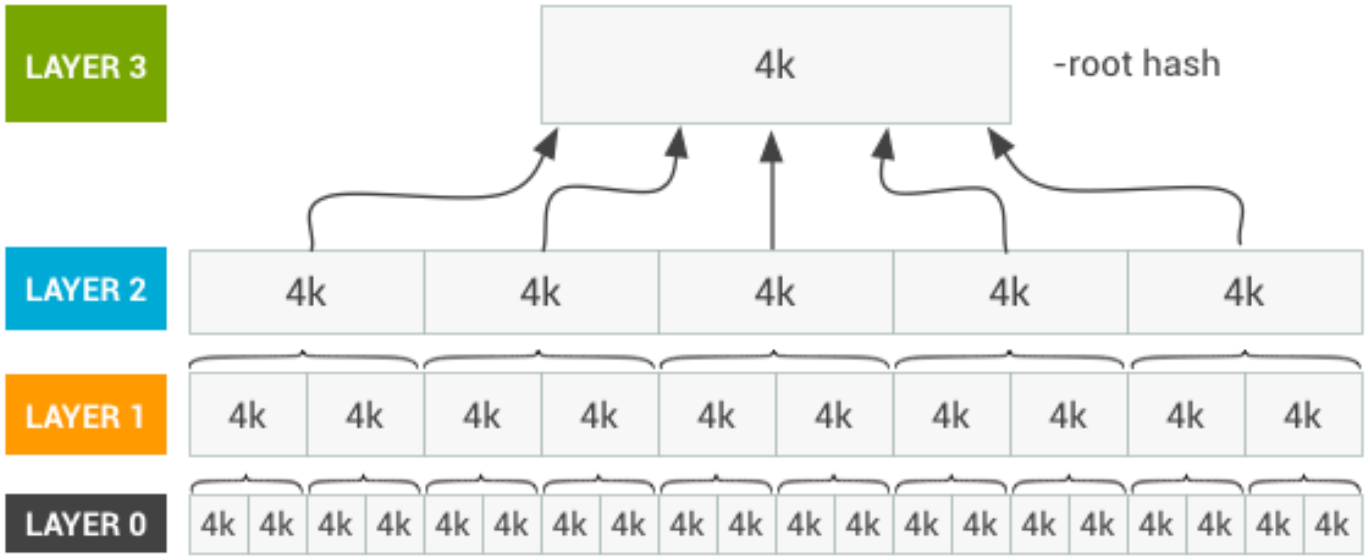


arm

How does DM-Verity work?

DM-verity uses a Merkle tree of storage blocks to protect the integrity of the read only data on the storage device, in a way that the integrity can be evaluated in a lazy fashion during runtime instead of pre-mount time.

It needs a singular trusted root hash to achieve security



Another type of Merkle tree

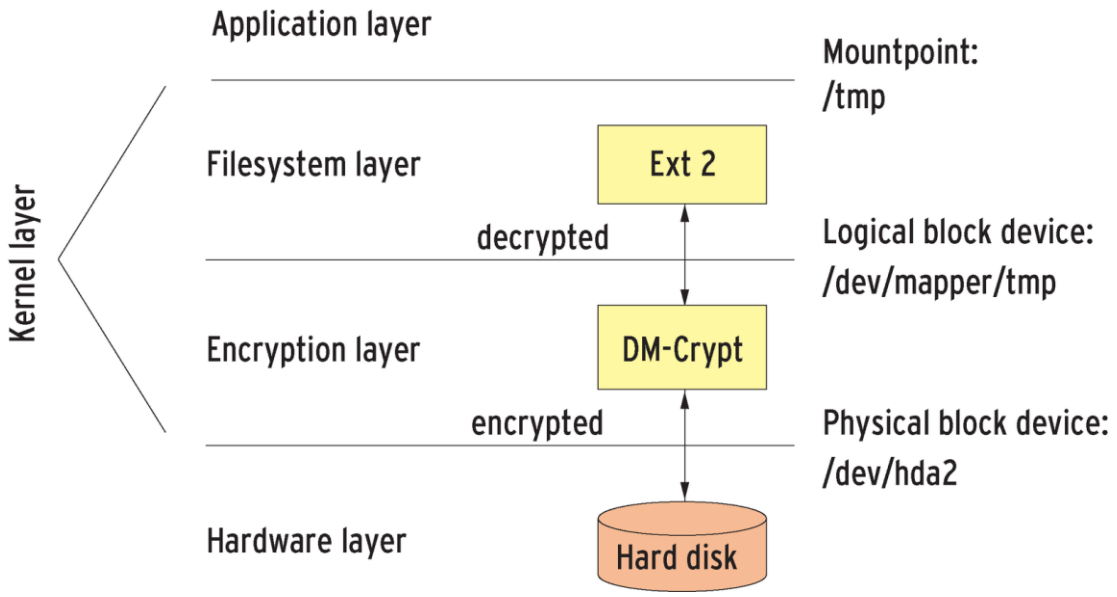
arm

Simple dm-verity setup example

```
# veritysetup format filesystem.img signature.img
# veritysetup create vroot fs.img sig.img \
ffa0a985fd78462d95c9b1ae7c9e49…10e4700058b8ed28
# mount -t ext2 /dev/mapper/vroot /media/
# umount /media
# veritysetup remove vroot
```

This examples uses SHA 256 to protect the integrity of the loopback file system provided the root hash is secure.

Full Disk Encryption (A.K.A DM-Crypt)



Transparent whole disk (read: partition) encryption scheme in device mapper.

Support advanced modes of operation, such as XTS, LRW and ESSIV.

It is used by Android devices to implement Full Disk Encryption.

Can also supply data integrity if used in conjunction with DM-Integrity and an AEAD.

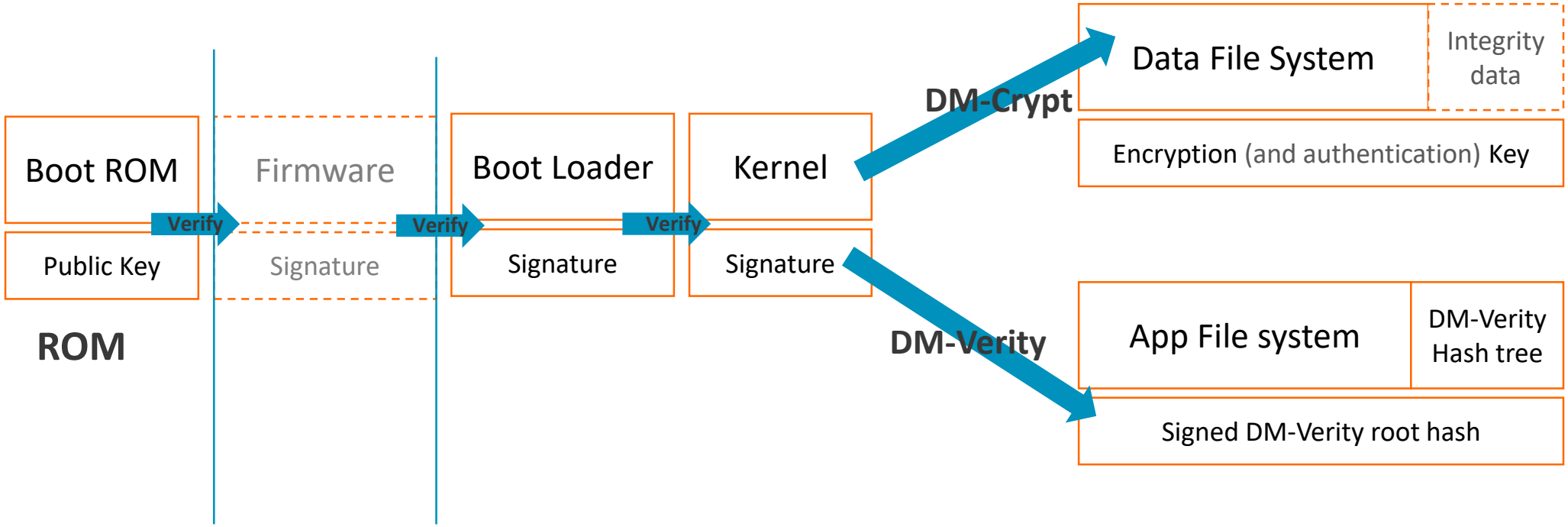
arm

Simple DM-Crypt setup example

```
# cryptsetup luksFormat fs3.img
# cryptsetup open --type luks fs3.img croot
# mke2fs /dev/mapper/croot
# mount -t ext2 /dev/mapper/croot /media
# umount /media/
# cryptsetup close croot
```

This examples uses AES in XTS operating mode to protect a loopback mounted partition file.

Example Simple System Setup



- Each stage verifies the next stage
- The public key may actually be a hash of the public key supplied later in the chain.
- There may be a chain of public keys, each signed by the previous (Chip → SoC → OEM)
- There may be multiple kernel and app file systems for ease of upgrade.

arm

File System Based Encryption (fscrypt)

File system based encryption provides encryption at the file system level, as opposed to block level of DM-Crypt.

Each directory may be separately and optionally encrypted with a different key.

Currently supported by the EXT4, UBIFS and F2FS file systems.

Allows multi level, multi user based protection, e.g. solution to Android “alarm clock” problem.

Designed to protect against “Occasional temporary offline compromise of the block device content, where loss of confidentiality of some file metadata, including the file sizes, and permissions, is tolerable.”*

- Currently, file data and file names are encrypted but other file meta data (e.g. size and permissions) are not.

* From: “EXT4 Encryption” by Mike Halcrow & Ted Ts’o presentation; http://kernsec.org/files/lss2014/Halcrow_EXT4_Encryption.pdf

arm

File system based encryption setup example

 # Make a random 512-bit key and store it in a file

> dd if=/dev/urandom of=key.data count=64 bs=1

 # Get the descriptor for the key

> ./fscryptctl get_descriptor < key.data

cd8c77009a9a3e6d

 # Insert the key into the keyring (using legacy ext4 options)

> ./fscryptctl insert_key --ext4 < key.data

cd8c77009a9a3e6d

> keyctl show

Session Keyring

827244259 --alswrv 416424 65534 keyring: _uid_ses.416424

111054036 --alswrv 416424 65534 _ keyring: _uid.416424

227138126 --alsw-v 416424 5000 _ logon: ext4:cd8c77009a9a3e6d

 # Make a test directory on a filesystem that supports encryption

> mkdir /mnt/disks/encrypted/test

 # Setup an encryption policy on that directory

> ./fscryptctl set_policy cd8c77009a9a3e6d /mnt/disks/encrypted/test

> ./fscryptctl get_policy /mnt/disks/encrypted/test

Encryption policy for /mnt/disks/encrypted/test:

Policy Version: 0

Key Descriptor: cd8c77009a9a3e6d

Contents: AES-256-XTS

Filenames: AES-256-CTS

Padding: 32

arm

File system based encryption usage example

 # Now we can make the file and write data to it

> echo "Hello World!" > /mnt/disks/encrypted/test/foo.txt

> ls -lA /mnt/disks/encrypted/test/

total 4

-rw-rw-r-- 1 joerichey joerichey 12 Mar 30 20:00 foo.txt

> cat /mnt/disks/encrypted/test/foo.txt

Hello World!

 # Now we remove the key, remount the filesystem, and see the encrypted data

> keyctl show

Session Keyring

1047869403 --alswrv 1001 1002 keyring: _ses

 967765418 --alswrv 1001 65534 _ keyring: _uid.1001

1009690551 --alsw-v 1001 1002 _ logon: ext4:cd8c77009a9a3e6d

> keyctl unlink 1009690551

1 links removed

> umount /mnt/disks/encrypted

> mount /mnt/disks/encrypted

> ls -lA /mnt/disks/encrypted/test/

total 4

-rw-rw-r-- 1 joerichey joerichey 13 Mar 30 20:00 wnJP+VX33Y6OSbN08+,jtQXK9yMHm8CFcI64CxDFPxL

> cat /mnt/disks/encrypted/test/wnJP+VX33Y6OSbN08+,jtQXK9yMHm8CFcI64CxDFPxL

cat: /mnt/disks/encrypted/test/wnJP+VX33Y6OSbN08+,jtQXK9yMHm8CFcI64CxDFPxL: Required key not available

arm

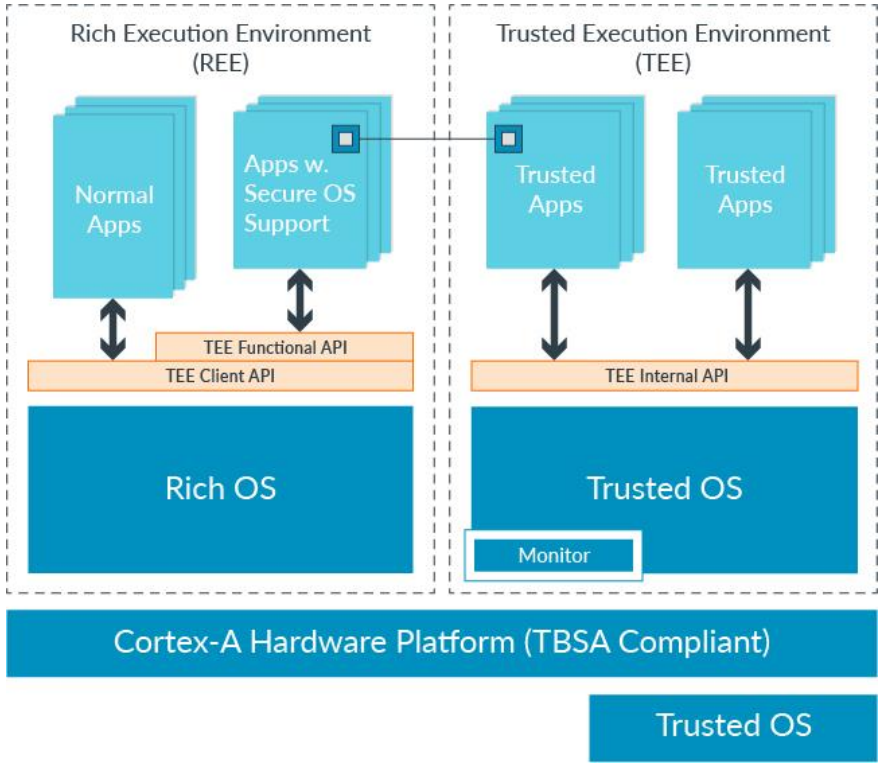
Trusted Execution Environment

The TEE is an isolated environment that runs in parallel with the operating system, providing security for the rich environment.

A TEE relies on hardware isolation mechanisms, such as Arm's TrustZone, Intel SGX or AMD SEE.

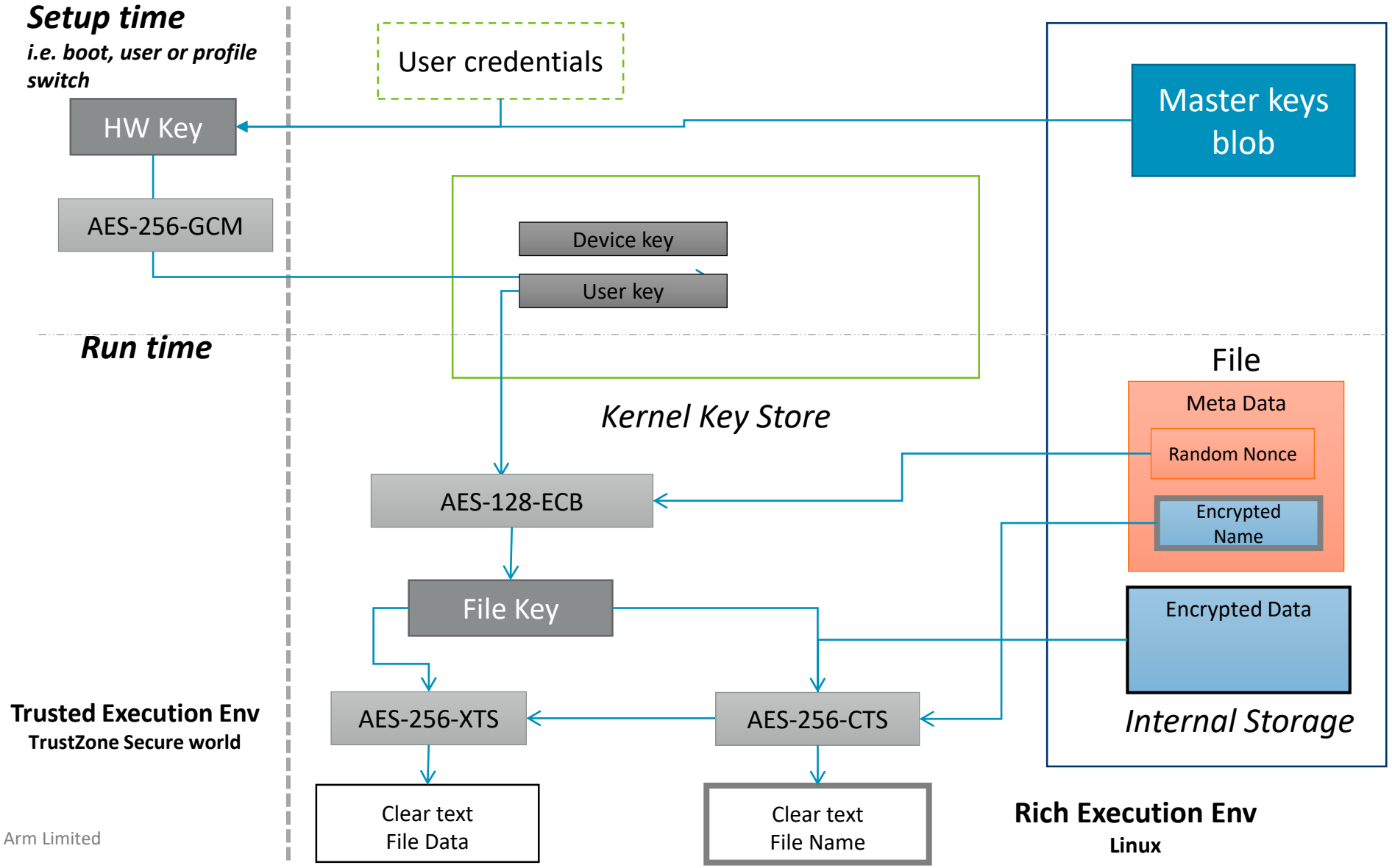
The TEE is intended to be more secure than the user-facing OS, dubbed Rich Execution Environment due to smaller attack surface, although this is not always the case.

TEE is supposed to run software that interacts with secrets (such as encryption keys) and provide services to the normal OS without the secrets being available.



arm

Android File Based Encryption (based on fscrypt)



arm

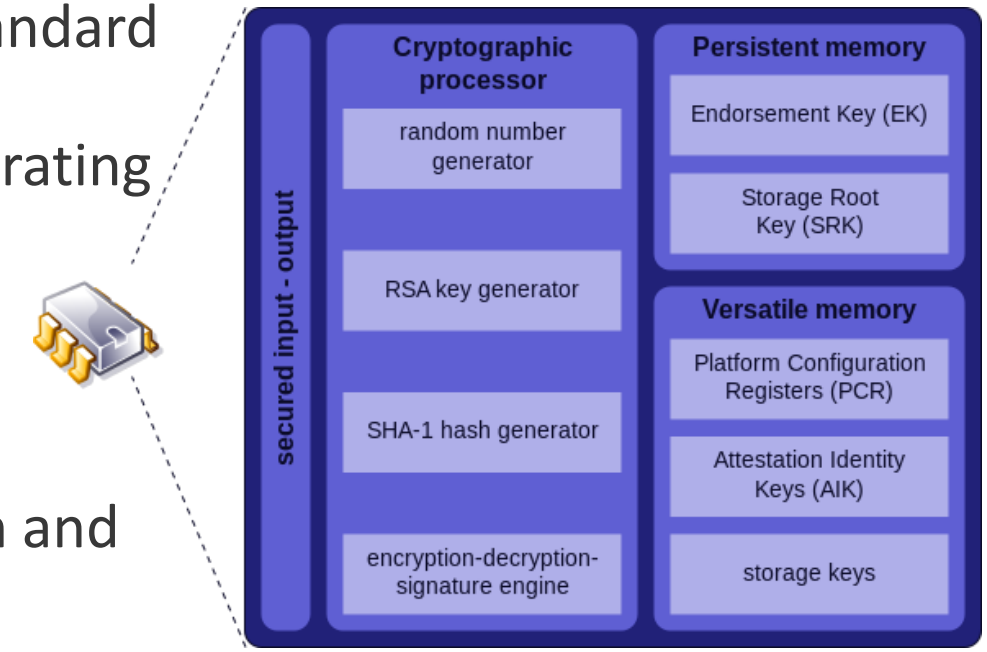
Trusted Platform Module

Trusted Platform Module (TPM) is an international standard for a secure crypto processor, which is a dedicated microcontroller designed to secure hardware by integrating cryptographic keys into devices.

TPMs offers facilities for the secure generation of cryptographic keys, and limitation of their use

It also includes capabilities such as remote attestation and sealed storage:

- Remote attestation – creates hashes of the hardware and software configuration.
- Binding – encrypts data using TPM specific key.
- Sealing – same as binding, but in addition specifies a state in which TPM must be, in order for the data to be decrypted (unsealed).



This figure was made by Guillaume Piolle. It is based on work by released by Everaldo Coelho and YellowIcon under the terms of the LGPL.

Source: https://en.wikipedia.org/wiki/Trusted_Platform_Module#/media/File:TPM.svg

arm

Integrity Measurement Architecture

Integrity Measurement Architecture, or IMA, is a Linux sub-system which provides runtime attestation services.

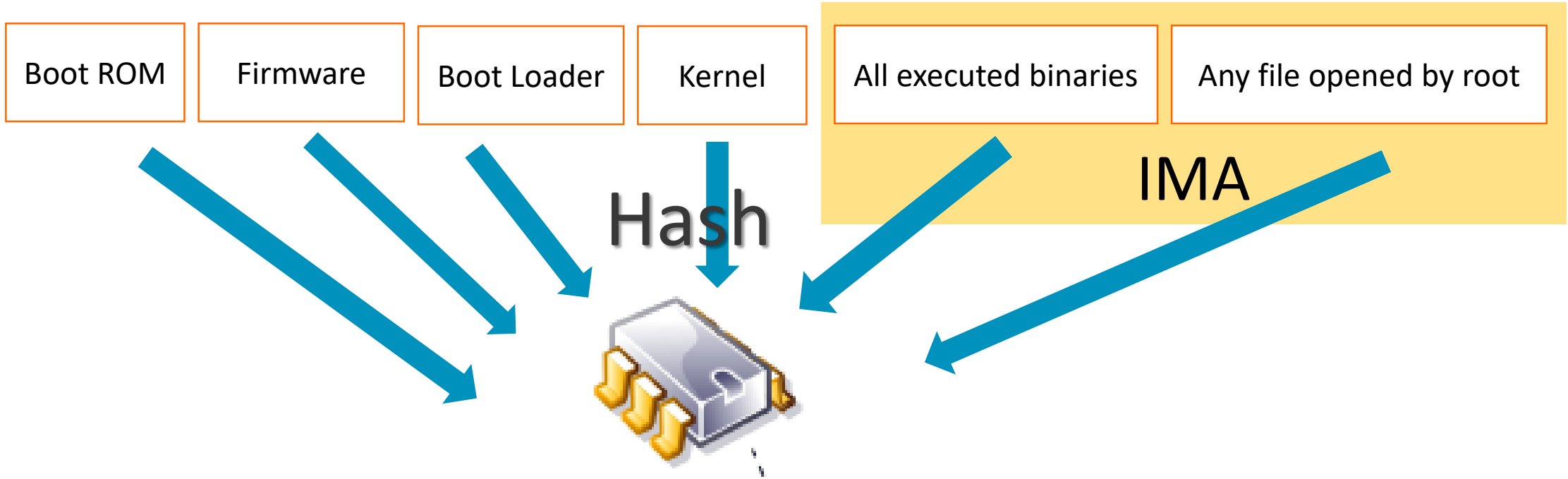
Working in conjunction with the TPM, it allows us to:

- **Measure** the system run-time state
- **Attest** to the system reliability, locally and remotely.
- **Allow** certain actions only if the state is as predicted.

"Attestation" simply means giving evidence of your reliability, just like the example above.

arm

How does IMA work?



IMA hashes all executables being run and all files opened by root and writes these hashes into the TPM PCR registers.

Together with similar service that do the same for hardware and software state during boot, we get a cryptographic hash representing the state of the whole system.

arm

How IMA can be used to protect a system?

This allows the TPM to only unseal (read: allow to use) certain keys if the system state captured by the TPM is as expected.

This allows local and remote attestation of the system state, that is take action only if the system is in a Known Good State™.

The IMA is further extended by a Linux Security Module known as EVM to allow certain Linux operations (e.g. running a specific binary) only if the TPM attested system state is as expected.

Why, I feel safe already ☺

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!

Gilad Ben-Yossef
gilad@benyossef.com
@giladby

arm