

TPM Is No Silver Bullet: Pitfalls in Embedded Device Security

David Gstir (david@sigma-star.at)

sigma star gmbh

2025-08-25

Hi!

David Gstir:

- › All things security @ sigma star gmbh
- › Security audits of basically everything that runs code
- › Engineering and consulting around Linux, embedded systems and security
- › Trainings



Intro

- › Customers from embedded sector approach us with various security requirements
- › TPM comes up more and more lately
- › Motivator often compliance with regulations (EU CRA, ISO 62443, ...), but also protecting IP and other reasons
- › Encountered some misconceptions ("*But we have a TPM! It'll solve this!*") about what a TPM will solve and what not
- › This talk is intended to be a guide for using TPM in embedded devices

Trusted Platform Module (TPM)

- › Dedicated security chip or software implementation (fTPM)
- › Open specification by Trusted Computing Group (TCG)
- › Connected via buses like SPI, I2C, LPC
- › Commonly used in PCs for long time now - required by Win11
- › Rather seldom use on embedded systems
- › Latest spec: TPM 2.0



Figure 1: Source: infineon.com

Core TPM Features

- › Cryptographic algorithms (ECC, RSA, HMAC, SHA-256, AES, ...)
 - › Also some weak ones you should not use anymore!
- › CSPRNG
- › Multiple Key hierarchies (Platform, Storage, Endorsement, NULL)
- › Authorization
- › Platform Configuration Registers (PCRs)
- › NVRAM storage
- › ...

Common Uses of TPMs

Secret Storage

- › Used for disk encryption with LUKS/dm-crypt (e.g. via systemd integration)
 - › Have disk encryption key bound to TPM
 - › Optionally, provide low entropy password (PIN) to unlock
- › Also private keys for TLS, VPN, SSH, ...
- › Key hierarchies for protected keys which cannot leave TPM
- › We can only *use* them by talking to TPM (if proper authorization is done)
- › Protect data using internal key (*binding*)
- › Small internal storage, but also storage on external media as protected blobs
- › A client passes the blob to the TPM to unbind it, which verifies and decrypts it

Measured Boot

- › Not to be confused with verified boot where you verify cryptographic signatures of next boot stage
- › Boot components (UEFI, bootloader, kernel, etc.) produce measurements on TPM
- › Extend (otherwise immutable) *PCRs* which contain cryptographic hash
- › Component change results in different PCR hash -> detectable modification
- › *Sealing*: TPM keys can be *tied to specific PCR values*, preventing their use if the boot state is altered. Combines nicely with secret storage

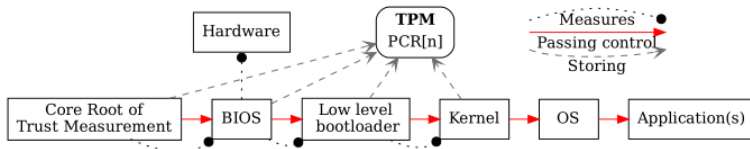


Figure 2: Source: Securing and Hardening Embedded Linux Devices by Marcin Bajer

(Remote) Attestation

- › Ensures the device is in a known, secure state before a remote party interacts with it
- › Provides a cryptographic proof of the system's boot and configuration state
- › Involves an attestation key (AK) and a trust chain to verify the TPM's identity
- › Result is a signed *quote* from the TPM, containing the current PCR values and a signature
- › Can be used to prove that a device is running a specific, untampered firmware version
- › Great overview by Matthew Garrett at *linux.conf.au*¹

¹<https://www.youtube.com/watch?v=FobfM9S9xSI>

TPM on Embedded

- › Not common, but possible
- › Often Arm-based, so UEFI parts needs some special treatment (measurements!)
- › U-Boot implements UEFI and can run UEFI applications²
- › Verified boot required as well: at least from vendor boot ROM to bootloader
- › fTPM can be run as TA via OP-TEE in Arm TrustZone (depends on security/certification requirements)
- › Great talk on this by Manuel Traut at *All Systems Go!* 2024³

²<https://docs.u-boot.org/en/latest/develop/uefi/uefi.html>

³<https://www.youtube.com/watch?app=desktop&v=eJihjE1zb1M>

Threat Model: PC vs. Embedded Device

- › User wants to protect their data vs. producer wants to protect their IP
- › Usually no physical presence (e.g. PIN entry) possible
- › Exposure to attacker is trivial: user is potential attacker
- › Device is often up 24/7
- › More hostile environment: easy tampering with hardware during normal operation
- › Longer lifecycles (10+ years)
- › Upside: Embedded software more tailored to single use case and thus “easier” to lock down

Pitfalls & Common Threats

Performance & Cold Boot Attacks

- › Cryptography on TPM is slow
- › Good enough to encrypt/decrypt asym. key, but not suitable for high volume traffic (e.g. symmetric crypto of VPN or TLS)
- › When performance is needed CPU has to be used
- › TPM will hand decrypted data back to CPU
- › Exposes secrets to main memory
- › Dedicated measures against cold boot attacks and similar attack vectors are needed!

Runtime Integrity Protection

- › Measured boot and verified boot protect boot chain only
- › Runtime modification often not covered (no PCR changes)
- › Code execution vulnerability can result in access to TPM
- › TPM alone cannot solve this
- › Hardening for disk encryption: seal key to PCRs and modify a PCR after disk is mounted
- › Additional measures:
 - › limited privileges per process
 - › FS permissions
 - › SELinux
 - › IMA/EVM
 - › Unique keys per device

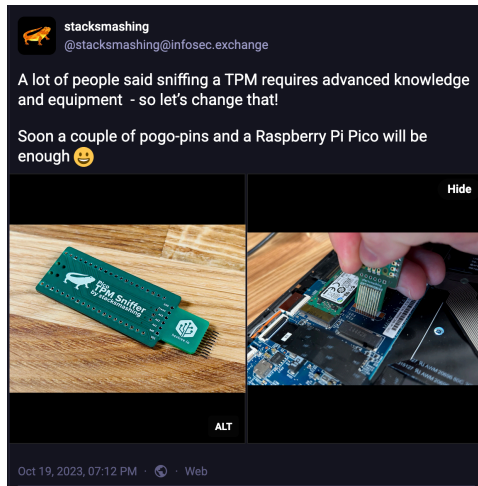
Direct Physical Attacks

- › Physical chip can be attacked using variety of physical attacks (tampering, side-channel, fault injection)
- › Measures against this depend on individual chip
- › Often very limited protection (chip coating, tamper evidence)
- › Often just compliance theater
- › Have a look at FIPS 140-2 or ISO/IEC 19790 security level certification of your chip
- › But there are easier ways...

Bus Snooping

- › Attach probes to physical bus between TPM and CPU to read messages
- › Is easier than it sounds⁴
- › TPM mitigations: session-based command and response parameter encryption
- › Multiple modes: HMAC-based can include PCRs
- › Client needs to opt in to these, otherwise plaintext comm is used!
- › Not all TPM clients do
- › systemd's TPM+dm-crypt integration does this already

⁴<https://github.com/nccgroup/TPMGenie>



Active Interposer Attacks

- › Can launch man-in-the-middle attacks between CPU and TPM
- › If encrypted & authenticated sessions are properly used, this can be partially mitigated
- › E.g. systemd's dm-crypt integration places Storage Root Key (SRK) metadata into LUKS header⁵
 - › MitM attacker will not have access to private SRK
- › Problem still: interposer can reset TPM and rebuild PCRs as needed
- › Kernel recently (6.10) gained ability to detect this
- › Still requires userspace to do full attestation of TPM's EK certificate
- › See talk by James Bottomley at FOSDEM 2025⁶

⁵<https://github.com/systemd/systemd/issues/22637>

⁶<https://fosdem.org/2025/schedule/event/fosdem-2025-4827-recent-tpm-security-enhancements-to-the-linux-kernel/>

TPM Hardware & Firmware Flaws

- › TPM firmware can contain flaws
- › Has happened before with ST chips: see TPM-FAIL (CVE-2019-16863)⁷
- › When fixable by update, you as vendor are responsible to ship the update!
- › Hardware flaws often impossible to fix by update
- › Also happened before: ROCA vuln. (CVE-2017-15361) in Infineon chips⁸
- › Measures against such flaws are limited
- › Additional hardening of whole system (OS, backend) can help (unique key per device, mitigate code execution flaws)

⁷<https://tpm.fail/tpmfail.pdf>

⁸<https://blog.cr.yp.to/20171105-infineon.html>

Software-based TPM Issues

- › fTPM removes bus snooping issue
- › Still vulnerable to side-channel, timing leaks and similar flaws.
- › E.g. faulTPM vulnerability⁹
- › Adds more things to maintain and keep secure
- › fTPM in TrustZone:
 - › fTPM code needs to be kept up-to-date
 - › TrustZones have flaws as well: e.g. memory flaws¹⁰
- › fTPM in OP-TEE:
 - › requires secure storage layer itself
 - › requires some form of hardware support (eMMC RPMB)
 - › eMMC RPMB: don't hardcode key in your code

⁹<https://arxiv.org/abs/2304.14717>

¹⁰<https://link.springer.com/article/10.1007/s11416-021-00413-y>

Alternatives to TPMs

- › Vendor-specific solutions like NXP CAAM
- › Custom solutions based on Arm TrustZone or similar TEE
- › Dedicated security chips e.g. Apples Secure Enclave
- › Hardware-security modules (HSMs)

Summary

- › TPM is definitely an options for embedded devices
- › It enables reuse of existing features we use on PCs (e.g. disk encryption)
- › Threat model is different
- › TPM alone cannot solve everything
- › Additional security measures need to be added

FIN

Thank you!

Questions, Comments?

David Gstir

david@sigma-star.at

