BKK19-215
TPM in TEE
Joakim Bech
Agenda

- Motivation for the talk
- TPM primer
- Firmware TPM
- Open discussion - “BoF”
Motivation for the talk

- People asked questions about running a (virtual) TPM in TEE, which led to
  - TPM study
  - Exploration of existing firmware TPM implementations
  - Looking at trade-offs and benefits
  - Comparison of similar technologies (TEE APIs)
  - Use case study, i.e., who, how, why and where?

- All that → decision
  - Whether to lead, invest time and efforts in TEE based TPM solutions
TPM Primer

- TPM is a passive device
- The TPM itself cannot alter a boot sequence
- Typically soldered on to a (mother-)board or as a standalone module inserted in a TPM header
- It can protect **your** secrets from others!

**Use cases**
- Measured boot
- Key generation and key storage
- Signature generation and verification
- Remote attestation
- Rollback protection
- Binding data to a certain platform and state
- Audit of a log trail
- Direct Anonymous Attestation - DAA
- Device identification
TPM 1.2 vs 2.0

- **TPM 1.2**
  - Uses RSA2048 and SHA-1
  - Generates and use the Endorsement Key (EK) and Storage Root Key (SRK) as main root keys (permanently stored in the TPM)
- SHA-1 has been deprecated by NIST for use in crypto operations since 2014
- This more or less makes TPM 1.2 devices “unusable”

- **TPM 2.0** support many more cryptographic algorithms (algorithm agility)
  - Enables more key sizes
  - Enables more hash functions
  - Random seeds as root of trust
  - Has four hierarchies, endorsement-, storage-, platform- and the NULL hierarchy
  - PCR’s a bit more flexible
  - + many more changes
Key generation

- In TPM 2.0 the keys are generated from a random seed
- You build your key hierarchy based on that
Platform Configuration Registers - PCR

- Typically around 24 PCR’s in a TPM
- Keep measurements for different parts of the system
- Holds a hash value
- Hash values are updated using an extend operation
  - $\text{Hash}_n = \text{Hash}(\text{Hash}_{n-1} || \text{new-data})$
- Starts from zero on boot
- Read PCR values by using signed “quotes”
Measured Boot

- Root of trust + chain of trust → computes a hash of the binary it is about to load
- Hash → TPM → updates the corresponding PCR’s
Remote attestation

- Remote party verifies the system state
- Verifies signed “quotes”
- One time or time based checking
Sealed keys and data

- Keys/data can only be unlocked when the system is in a certain state!
- Example: Only unlock keys at a certain boot stage (not before, after)

- Very powerful, but also very fragile! How to update software?
  - TPM 1.2: Very tricky!
  - TPM 2.0: Unseal $\rightarrow$ upgrade $\rightarrow$ sign new PCR value $\rightarrow$ reseal.
  - What if a hard drive starts to get unrecoverable bit errors?

- Bitlocker in Microsoft Windows uses this
  - TPM unwraps a FDE symmetric key
Authorization

- **Passwords**
- **HMAC**
  - Shared secret TPM ↔ Caller
  - More secure than plain passwords
- **Policy**
  - Correct sequence of commands
  - Most complex and most feature rich
- **Trail**
  - “Test authorization” → calculates future (good) PCR values.

Examples of Policy Authorization (*):

- Certain values in PCRs
- A password
- A sequence of TPM commands
- Physical presence
- A value or condition in NV Index

(*) Doesn’t have to be mutually exclusive
Resource Manager

- TPM can only hold three objects at a time
- Need a **Resource Manager** (RM) that handles and controls the communication with the TPM
- RM is defined in the TCG/TSS stack
- Used to be a pure user space component in Linux, but since Linux Kernel 4.12 there is a RM in the kernel itself
TSS - TCG Software Stack

Image Source: TSS_Ovewview_Common_Structures_Version-0.9_Revision-03_Review_030918.pdf
Firmware TPM

“Leveraging a TEE to implement TPM functionality”
Definition of firmware TPM

My definition:

- A TPM implementation running in some kind of (secure) firmware
- The implementation works transparently with the existing TSS stack
- As compatible with a regular TPM chip as possible
Microsoft fTPM

- TCG TPM 2.0 spec has lots of code written in C (main contributor is Microsoft)
- Microsoft took that code and created fTPM [1]
  - Official TCG reference implementation of the TPM 2.0 Specification
- Runs on x86 (Intel SGX) and Arm (TrustZone)
  - They’ve used OP-TEE for the Arm reference
- Shipped on millions of mobile devices running Windows!
- Changed semantics of some commands
  - Overcome Arm TrustZone limitations

- Talk from USENIX 2016 [2] goes more into the details

[1] https://github.com/Microsoft/ms-tpm-20-ref
Firmware TPM observations

Crypto operations
- TPM: Slow
- TEE: Fast

Random number generation
- TPM: Use internal TRNG (no requirements/specs to follow!)
- TEE: Can use HW or SW

Storage
- TPM: NV RAM (tiny)
- TEE: Regular flash, RPMB, ...

Secure clock
- TPM: Internal counter
- TEE: Should have a secure timer

Seeds
- TPM: never leaves the chip
- TEE: Use fuses? flash/RPMB?
  - Cannot re-create seeds stored in fuses!
  - Flash? No good for this
  - RPMB?

Side Channel Attacks?
TEE based fTPM takeaway

- Behavior same as a regular TPM
- Important to let users of fTPM know about the trade-offs and differences compared to a real TPM
- Need guidelines with best practices for provisioning etc
- Boot firmware needs to be changed
- “Linux” needs no changes (*) if we’re using the TSS stack

<table>
<thead>
<tr>
<th>Features</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>fTPM == TPM</td>
<td>fTPM != TPM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fTPM != TPM</td>
<td></td>
</tr>
</tbody>
</table>

(*) needs a TSS/TPM ↔ TEE driver
TPM in TEE discussions - “BoF”

“Why?”
“How?”
“Who?”
Why do we want to run a TPM in a TEE?

- BoM: Save a few cents by not using a TPM chip?
- Save some area on the PCB?
- Reusing existing API on devices with no TPM chip? Backward compatibility?
- Constraints on existing devices?
  - Mobile devices typically don’t have a TPM chip

- Why not simply implement the needed features directly in the TEE?
  - TPM comes with many features, it’s likely that we’re only interested in a subset
  - TSS stack is big and complicated
  - Other “TEE” specs already includes similar features (ex. attestation in OTrP)
How should a fTPM stack look?

- Our candidate is BL32 - OP-TEE
- How to deal with PCR measurements before BL32 has been loaded?
  - Store extended/running hash somewhere?
  - Hand over PCR hash to BL32 using some register?
  - Start PCR hashing from BL32? State/document that as a limitation/trade-off?

- Linux kernel interfaces?
  - Is there a need for something similar to vTPM proxy driver?
  - (MS has an OP-TEE/fTPM driver: https://github.com/ms-iot/linux)
Who is interested in this?

- It’s important to know this to justify working with this
- We’re not necessarily asking for company names
  - Think, SIGs, community projects etc that could benefit from this
- Are we aware of any stack that relies on TPM that would benefit by using a TEE based TPM?
What’s next

● Need to “nail” the use cases
● Define how new interfaces should look like
● Boot components: add communication support for TEE based TPM’s
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org