TEEP @ Hackathon

Hannes Tschofenig

(hannes.Tschofenig@arm.com)

Agenda

- What is TEEP?
  - History: TEEP protocol vs. OTrP
  - Architecture
- Goals and project ideas
- TrustZone Integration

TEEP - Trusted Execution Environment Provisioning

A software isolation technology

TEEP
Internet-Draft
Intended status: Informational
Expires: August 11, 2020

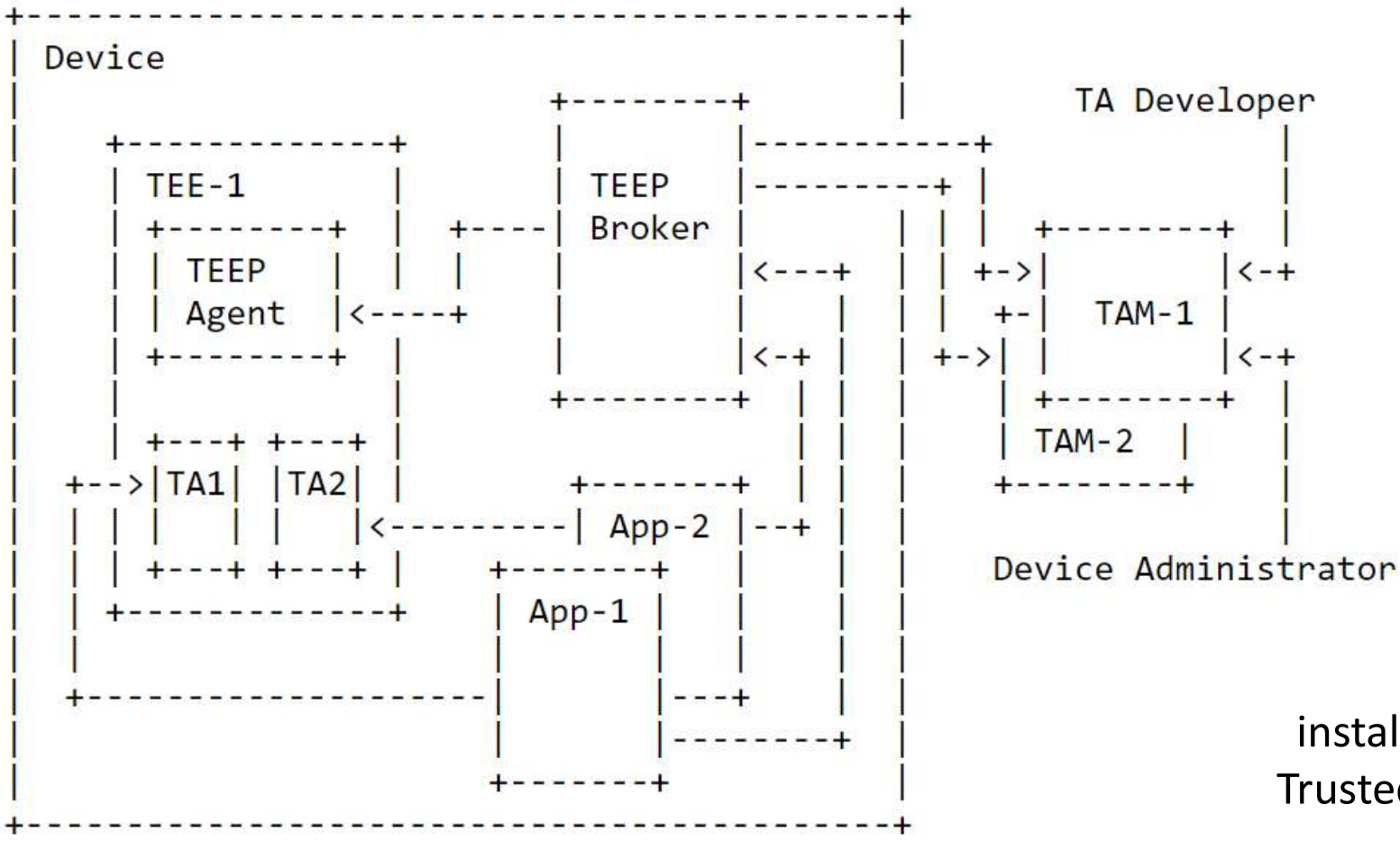
M. Pei
Symantec
H. Tschofenig
Arm Limited
D. Thaler
Microsoft
D. Wheeler
Intel
February 08, 2020

Trusted Execution Environment Provisioning (TEEP) Architecture
draft-ietf-teep-architecture-06

https://tools.ietf.org/html/draft-ietf-teep-architecture-06

The Trusted Execution Environment (TEE) concept is designed to execute applications in a protected environment that enforces that only authorized code can execute within that environment, and that any data used by such code cannot be read or tampered with by any code outside that environment, including by a commodity operating system (if present).

Architecture



The TEEP protocol installs, updates, and deletes Trusted Applications (TAs) in a device with a TEE.

TEEP Protocol vs. Open Trust Protocol (OTrP)

- OTrP was the proposed protocol solution submitted to the TEEP working group based on prior work done outside the IETF.
  - Expired draft here: https://tools.ietf.org/html/draft-ietf-teep-opentrustprotocol-03
  - Open source implementation exists: https://github.com/dthaler/OTrP
- TEEP working group generalized the protocol to focus on additional use cases, more TEEs, re-use ongoing IETF work and simplified the design.
- The result is the TEEP protocol replacing the OTrP protocol: https://tools.ietf.org/html/draft-ietf-teep-protocol-00
- Transport specified: https://tools.ietf.org/html/draft-ietf-teep-otrp-over-http-04

TEEP Protocol vs. Open Trust Protocol (OTrP)

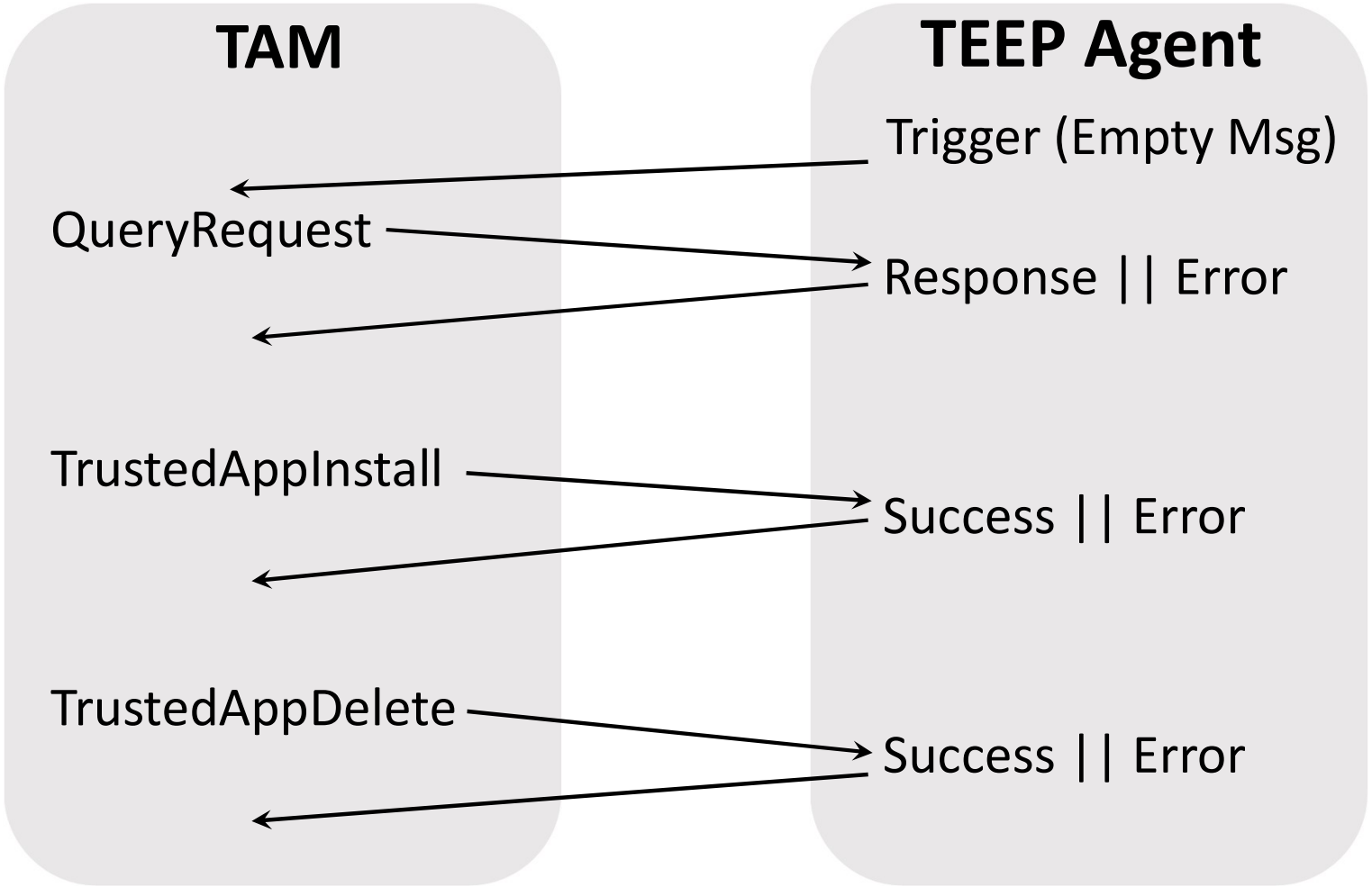
TEEP Protocol

- Uses CBOR and JSON encoding (with COSE and JOSE, respectively)
- Attestation based on RATS
- TA management based on SUIT
- Security Domain management removed from base protocol

OTrP

- Uses JSON and JOSE
- Attestation custom to OTrP
- TA management custom to OTrP
- Dropped key exchange for personalization data protection

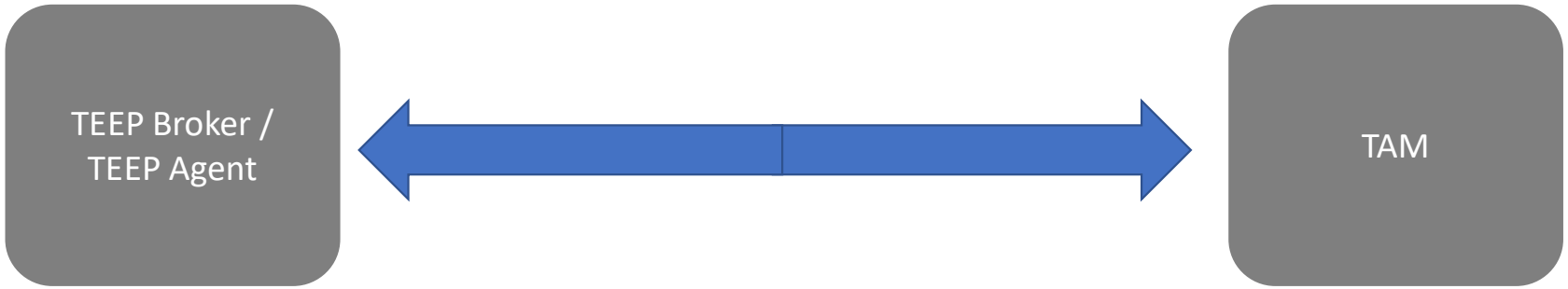
TEEP Protocol



Goals

- Verify TEEP protocol specification (readability, clarity, completeness)
- Add text for JSON/JSON spec to TEEP protocol specification
  (It is there via CDDL but more is needed to fully describe it.)
- Add examples (for both encodings)
- Learn from the integration into TrustZone and SGX.

Projects

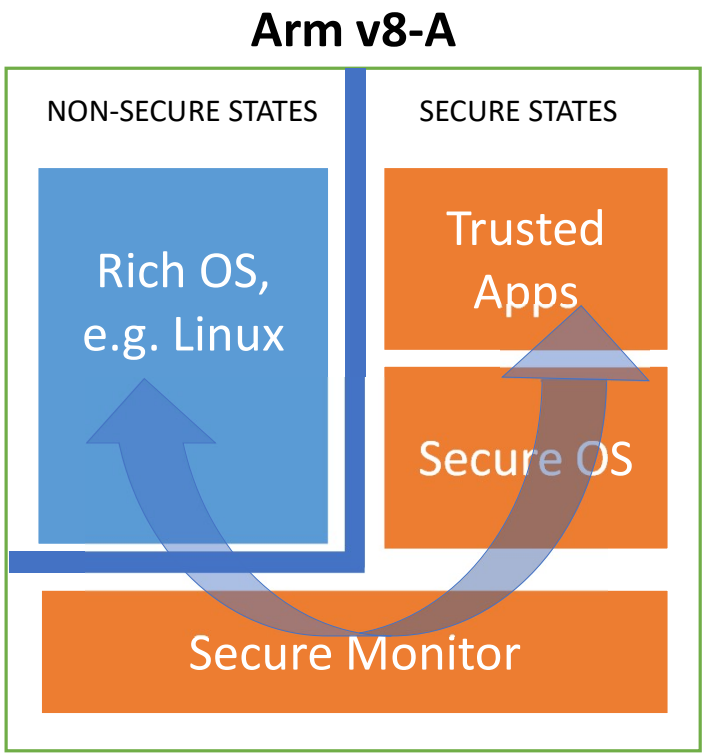


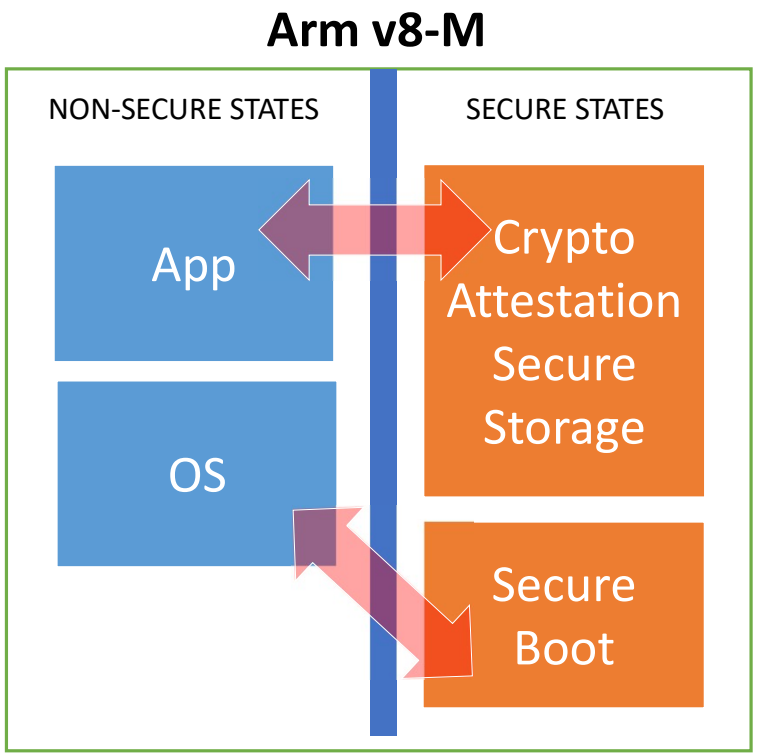
- Can we create a prototype implementation?
  - Client-side and server-side -- in 2 days? JSON/JOSE-based encoding – for example
  - Can we use different languages (Java/Python on TAM-side, and C on the client-side)
  - Can we re-purpose existing OTrP code (e.g., Dave's code) for TEEP?
  - Can we do some interop testing afterwards?
- Are we able to integrate SUIT and/or RATS?

Projects, cont.

- Could we even get the integration into TrustZone done?
- Note that there are two “types” of TrustZone:
  1. TrustZone for v8-M
  2. TrustZone for A-class

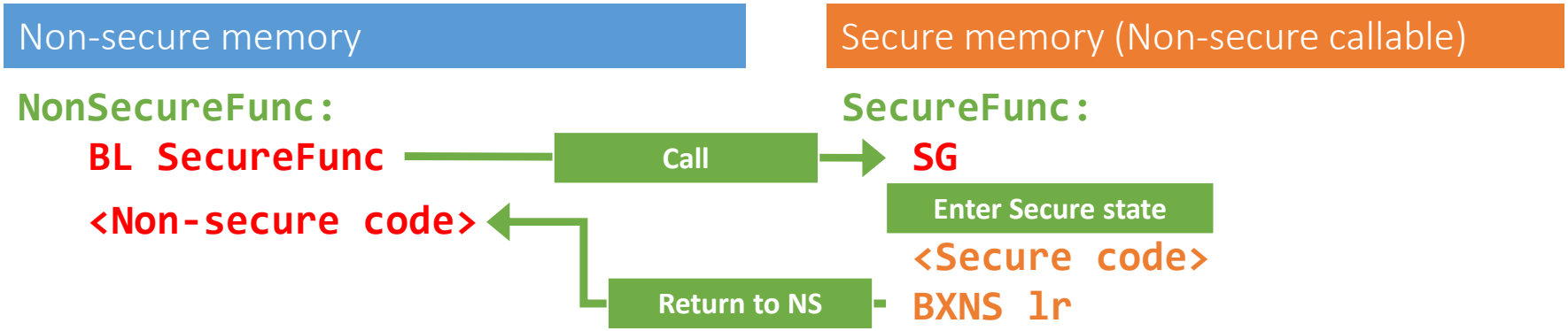
TrustZone





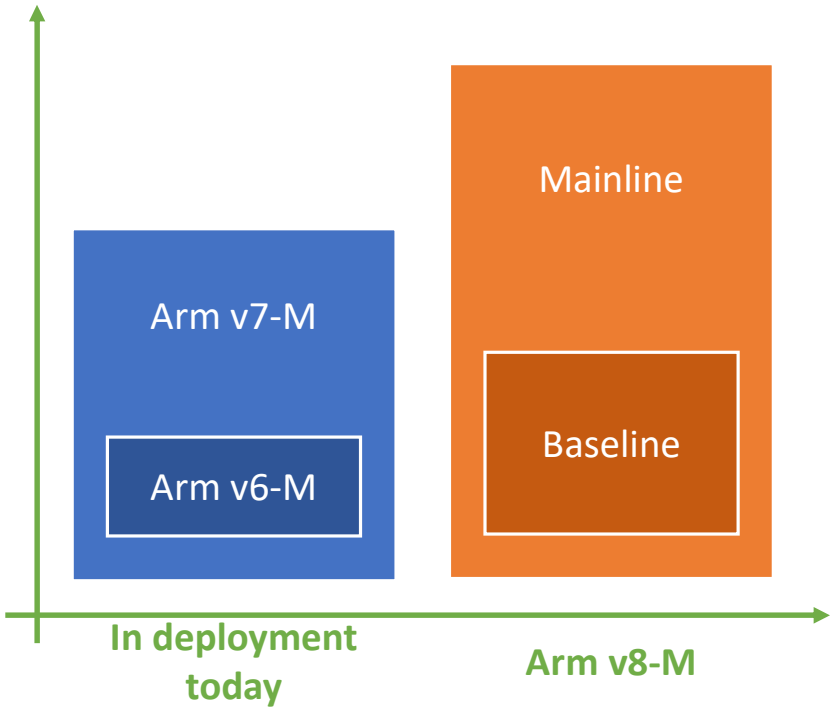
Secure transitions handled by the processor
to meet embedded system latency requirements

Cross-Domain Function Calls



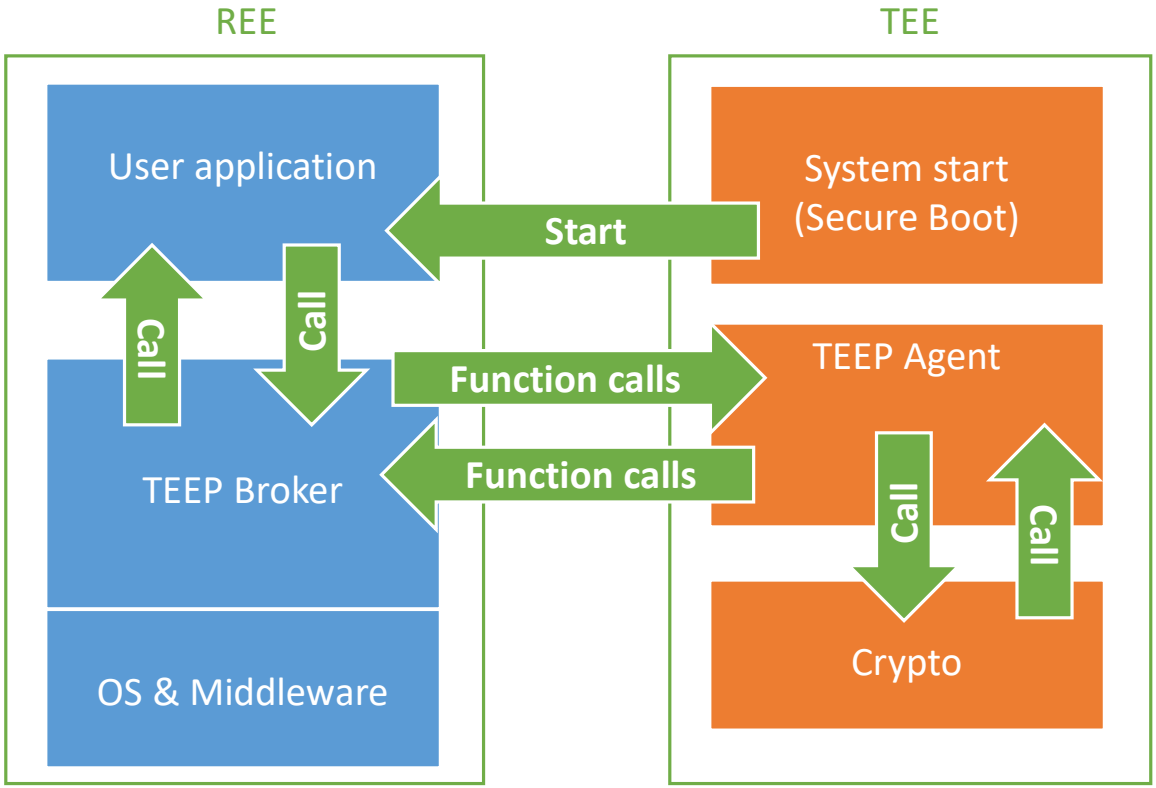
- Guard instruction (SG) polices entry point
  - Placed at the start of function callable from non-secure code.
- Non-secure → secure branch faults if SG isn't at target address
  - Can't branch into the middle of functions
  - Can't call internal functions.
- Code on Non-secure side identical to existing code.

ARMv8-M Sub-profiles



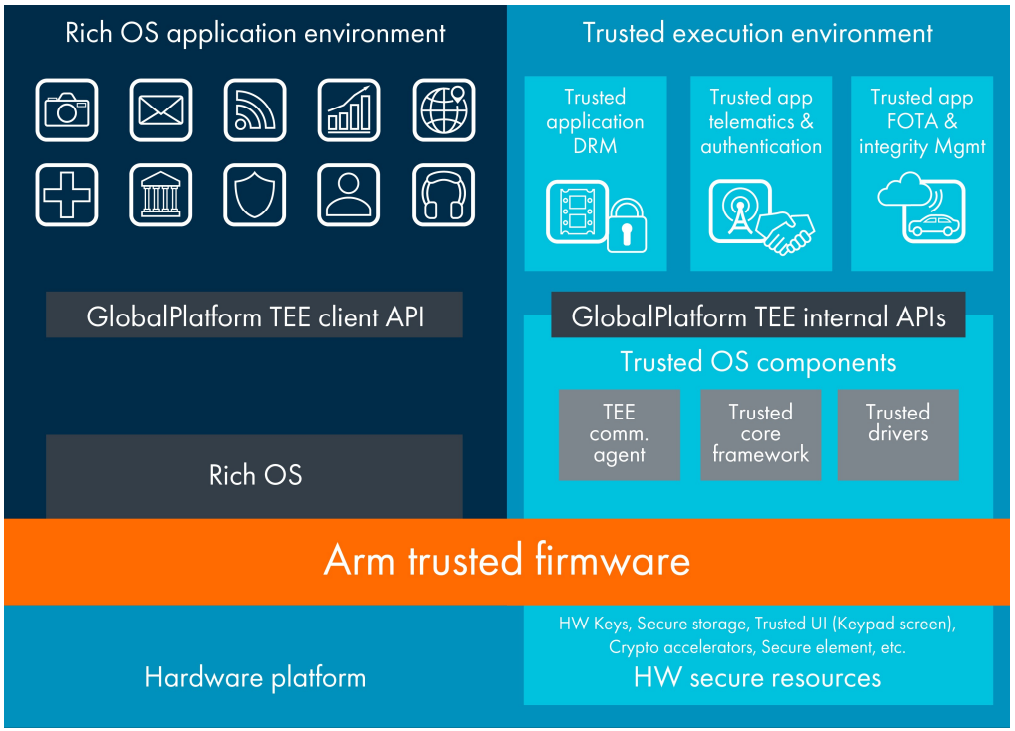
- Arm v8-M ***Baseline***
  - Lowest cost, and smallest implementations
  - Example: Cortex M23
- Arm v8-M ***Mainline***
  - For general purpose microcontroller products
  - Optional DSP, floating-point and ML extensions.
  - Examples: Cortex M33, Cortex M55 (Helium extensions)
- Variants with physical security properties available as well
  - Example: Cortex M35P

Possible Software Architecture



- Non-secure project cannot access Secure resources.
- Secure project can access everything.
- Secure side contains other security-relevant code besides TEEP, such as secure boot, attestation, crypto, secure storage, etc.

TrustZone for A-class



- GP specs:
  - https://globalplatform.org/specs-library/
- Reference implementation for monitor code: Arm Trusted Firmware for A class (TF-A)
  - https://www.trustedfirmware.org/
  - https://git.trustedfirmware.org/TF-A/
- Reference implementation for Trusted OS: OP-TEE
  - https://github.com/OP-TEE/
  - https://optee.readthedocs.io/en/latest/
  - https://github.com/linaro-swg/optee_examples

Communication

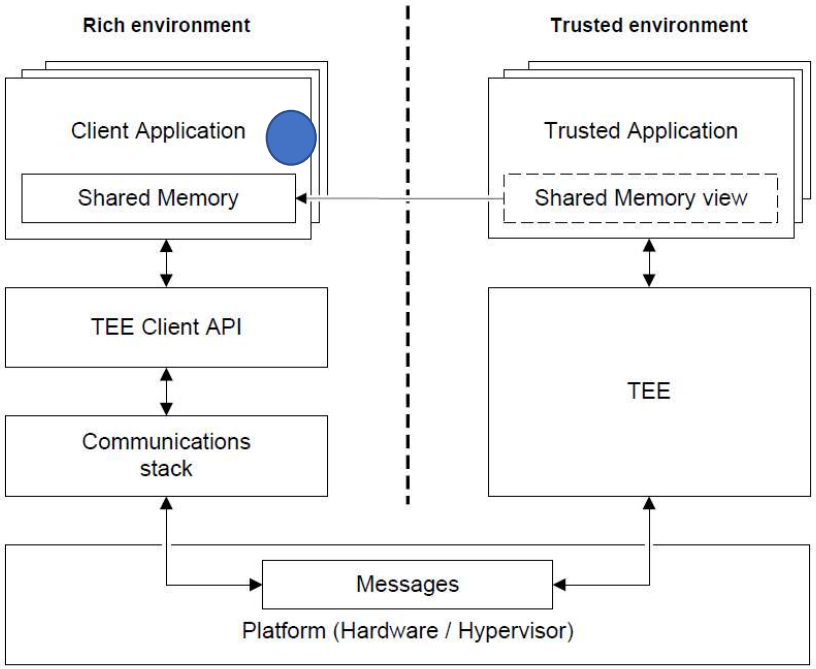


Figure 2-1: TEE Client API System Architecture

Reference: TEE Client API Specification - Version 1.0

1. TEEC_InitializeContext(ctx)
2. TEEC_OpenSession(ctx,session, UUID,…)
3. // create command structure
4. TEEC_InvokeCommand(session, cmd, ..)
5. TEEC_CloseSession(session)
6. TEEC_FinalizeContext(ctx)

Communication, cont.

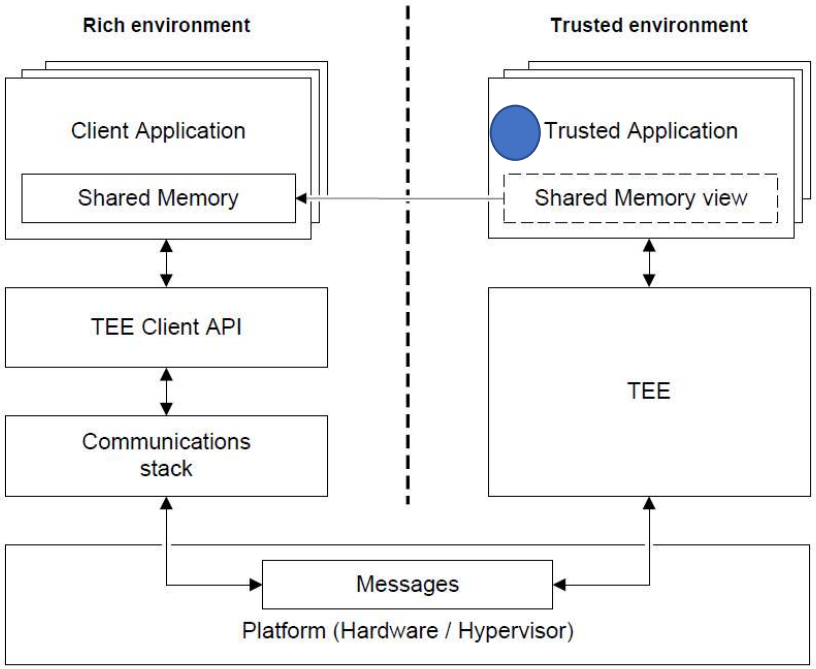


Figure 2-1: TEE Client API System Architecture

Reference: TEE Client API Specification - Version 1.0

- TA_CreateEntryPoint (..):
  - Called when the TA is created.
- TA_DestroyEntryPoint(..)
  - Called when the TA is destroyed.
- TA_OpenSessionEntryPoint(..):
  - Global initialization of the TA.
- TA_CloseSessionEntryPoint(..):
  - Called when the TA session is closed.
- TA_InvokeCommandEntryPoint (..): Calls functions based on the commands issued.

Communication
Passing short values

REE App

```
op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT, TEEC_NONE,
                                 TEEC_NONE, TEEC_NONE);
op.params[0].value.a = 42;
```

TA

```
uint32_t exp_param_types = TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INOUT,
                                           TEE_PARAM_TYPE_NONE,
                                           TEE_PARAM_TYPE_NONE,
                                           TEE_PARAM_TYPE_NONE);

if (param_types != exp_param_types)
        return TEE_ERROR_BAD_PARAMETERS;
params[0].value.a++;
```

Communication
Shared Memory

REE App

```
/* 1. Register the shared key */
op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
                                 TEEC_NONE, TEEC_NONE, TEEC_NONE);
op.params[0].tmpref.buffer = K;
op.params[0].tmpref.size = sizeof(K);

fprintf(stdout, "Register the shared key: %s\n", K);
res = TEEC_InvokeCommand(&sess, TA_HOTP_CMD_REGISTER_SHARED_KEY,
                         &op, &err_origin);
```

TA

```
uint32_t exp_param_types = TEE_PARAM_TYPES(TEE_PARAM_TYPE_MEMREF_INPUT,
                                           TEE_PARAM_TYPE_NONE,
                                           TEE_PARAM_TYPE_NONE,
                                           TEE_PARAM_TYPE_NONE);
```

If (…) …

```
memset(K, 0, sizeof(K));
memcpy(K, params[0].memref.buffer, params[0].memref.size);

K_len = params[0].memref.size;
```

Updating Code

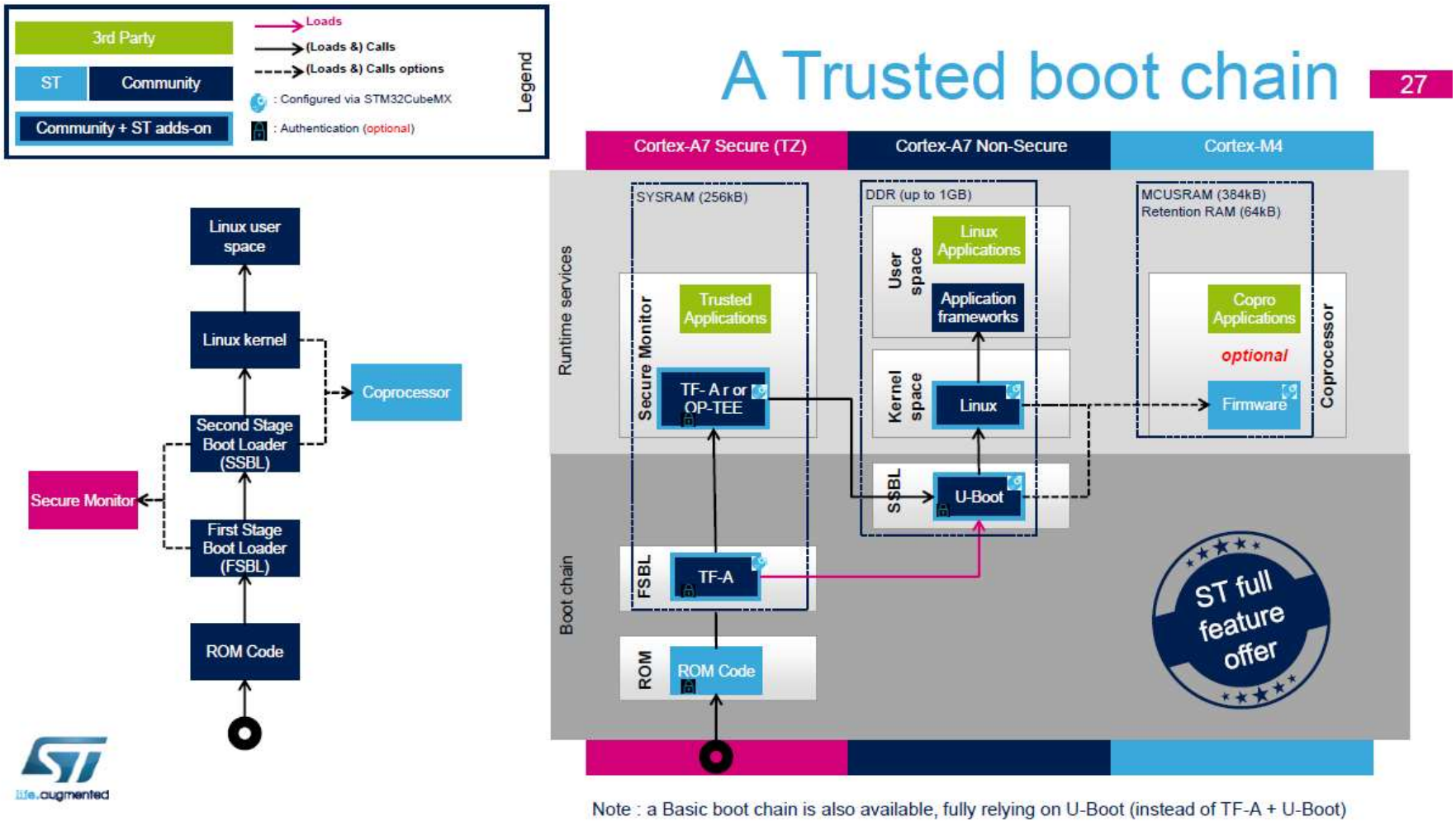


Figure copied from STM32MP1 documentation.

Summary

- For a TrustZone-based device, TEEP offers a protocol for managing the lifecycle of TAs (or code in general).
  - TEEP uses RATS and SUIT
- A non-TrustZone-based system may use TEEP for parameter negotiation
  - It may or may not use RATS in that case.
- RATS may be building block in a number of protocols where attestation functionality is desired.