Rust OP-TEE TrustZone SDK

Mingshen Sun

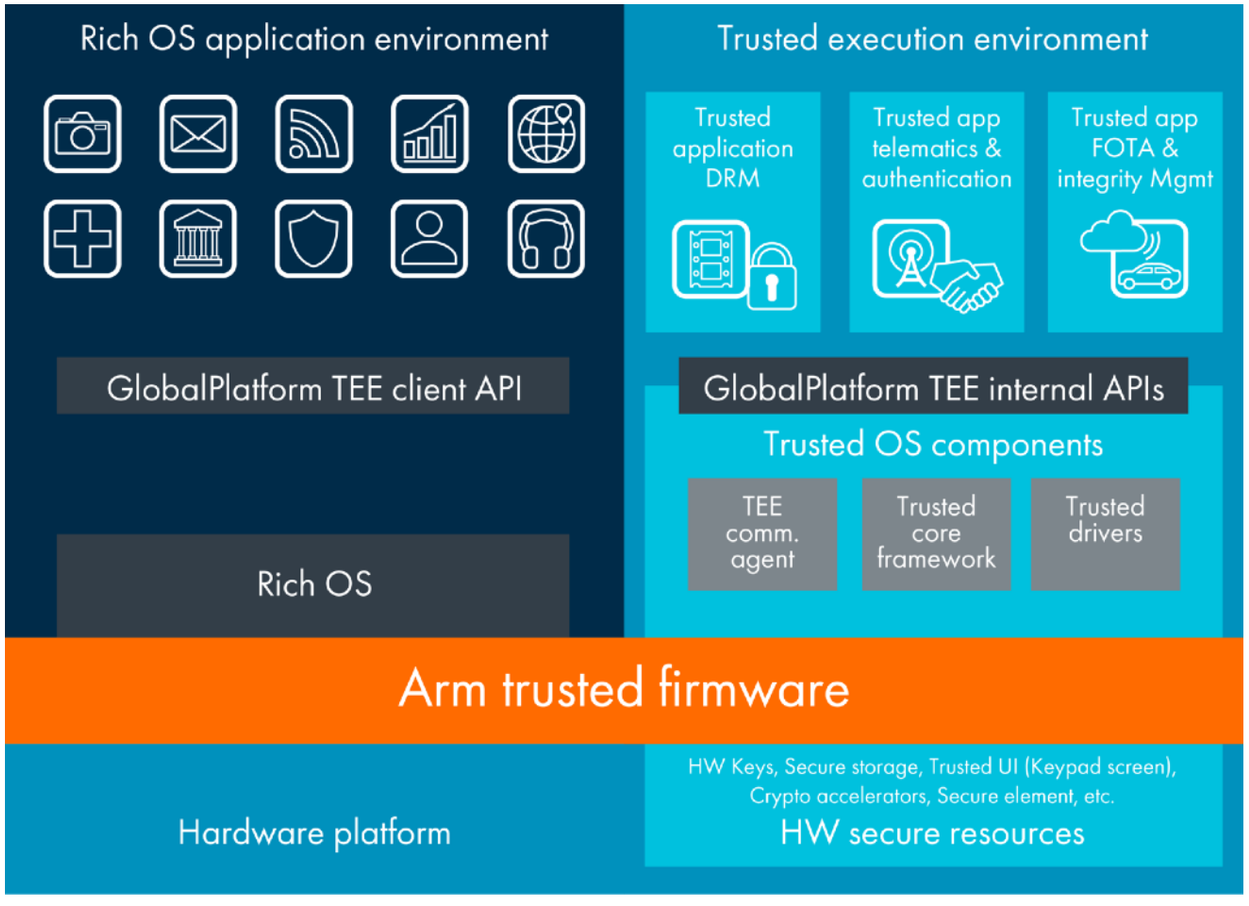
Baidu X-Lab

RustCon Asia, Beijing, April 2019

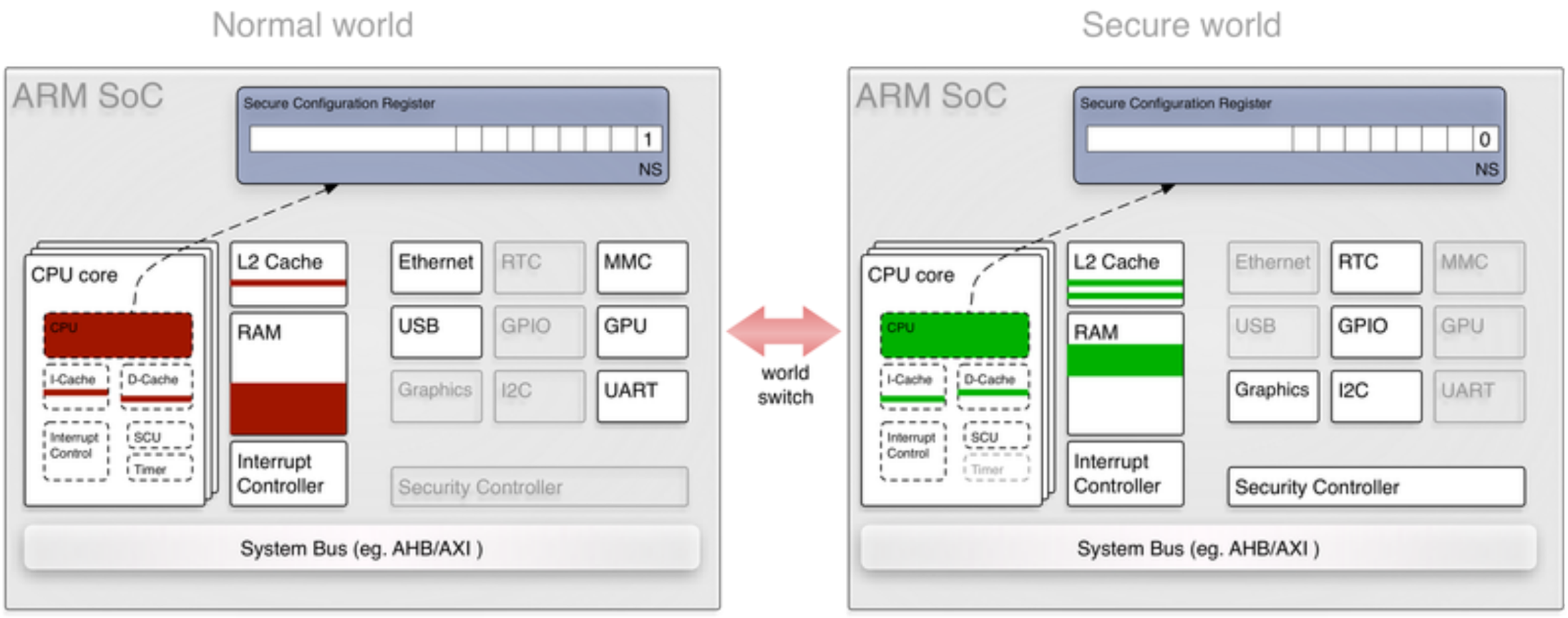
Background

- ARM TrustZone provide ***trusted execution environment*** in mobile phone and embedded devices

- TrustZone secures mobile payment, identification authentication, key management, AI models, DRM,OS integrity, etc.

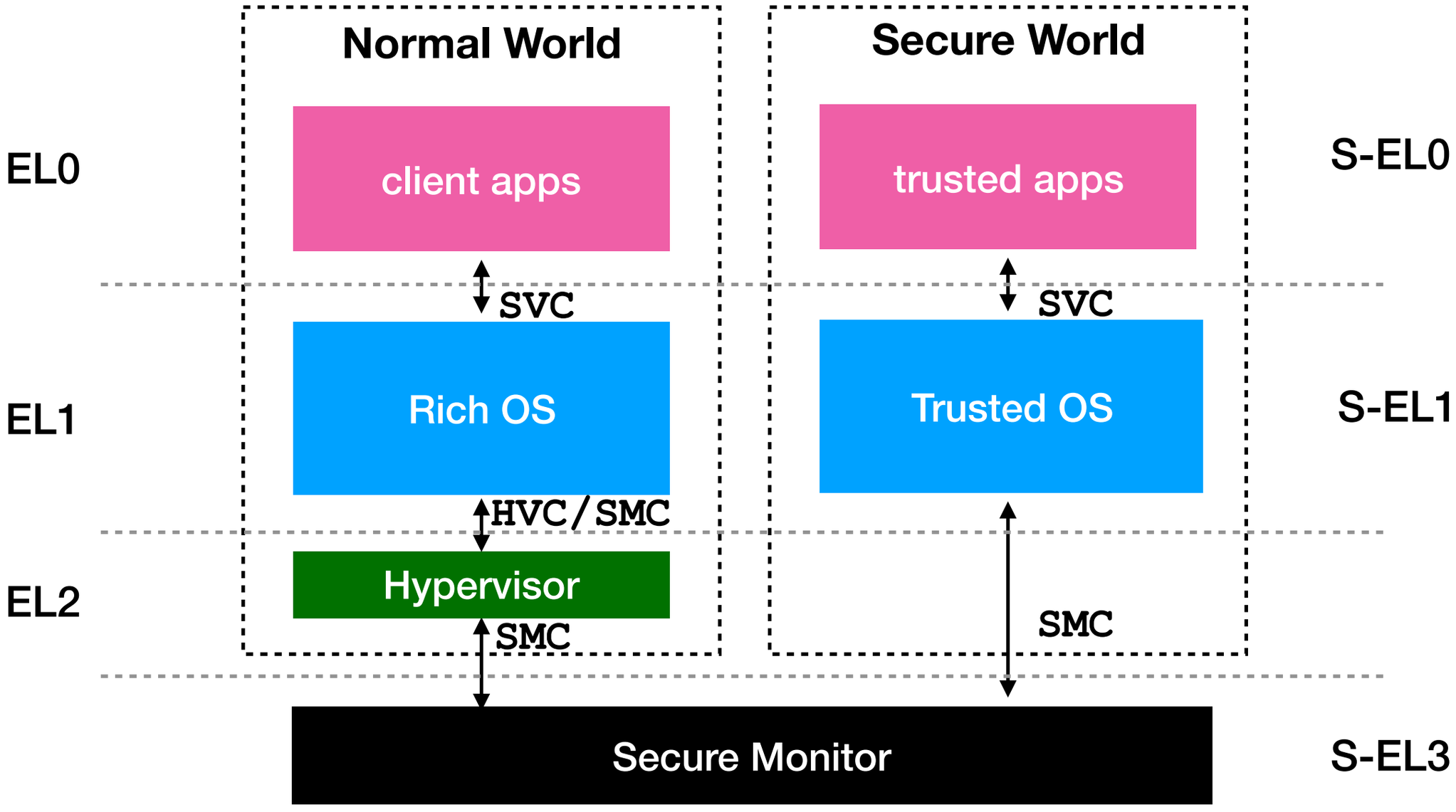


TrustZone Architecture



An Exploration of ARM TrustZone Technology: https://genode.org/documentation/articles/trustzone

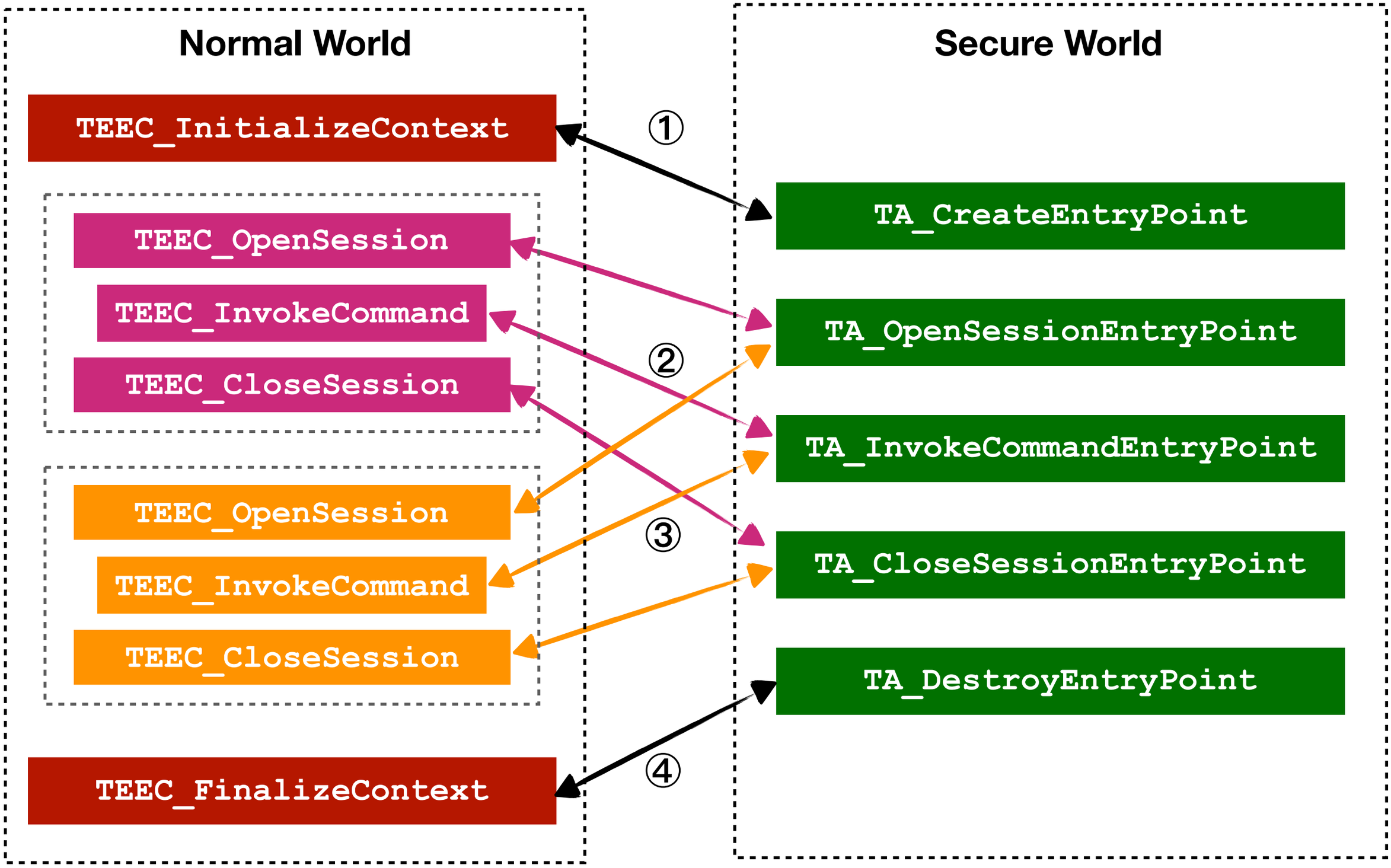
TrustZone Architecture



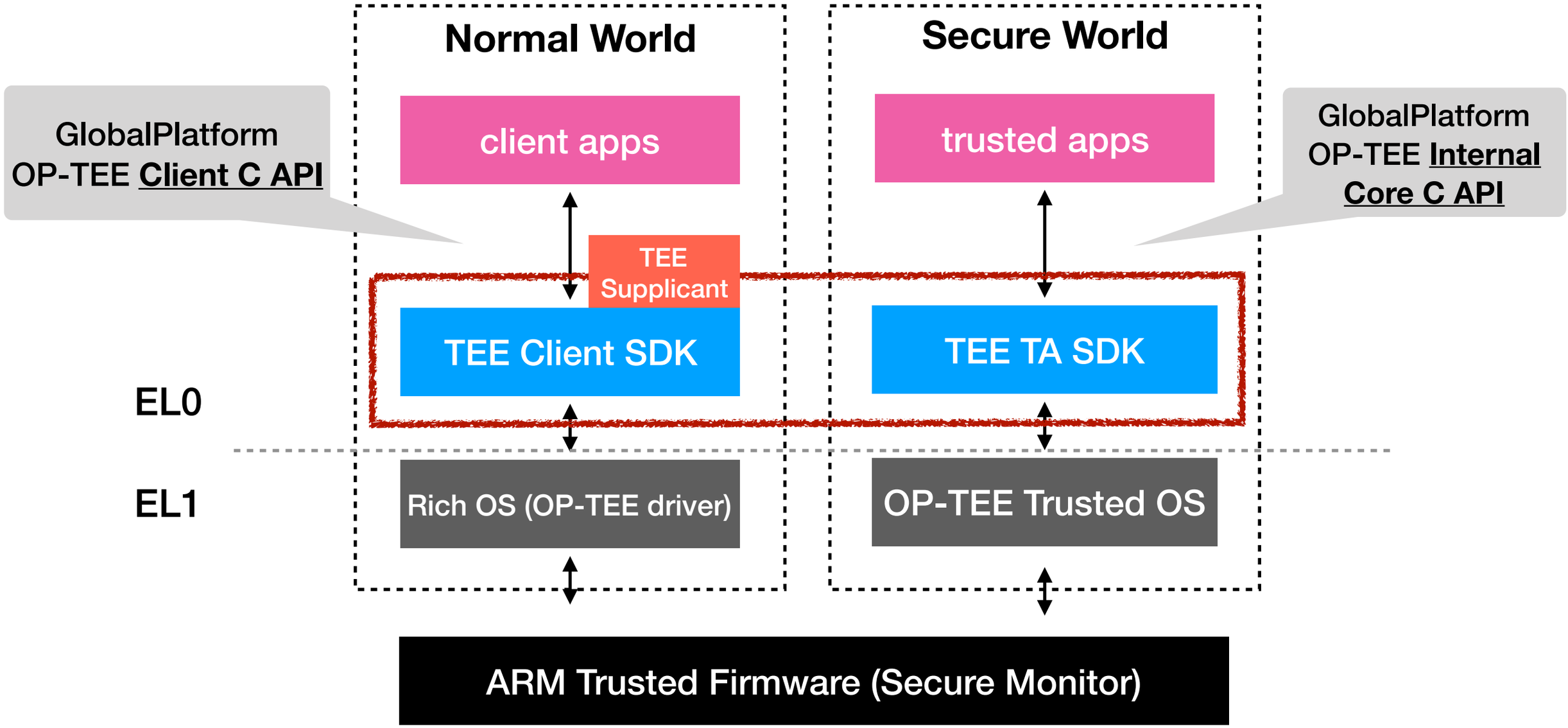
Background

- **GlobalPlatform** TEE specifications
  - *TEE System Architecture* (**`GPD_SPE_009`**): defines a general TEE architecture
  - *TEE Internal Core API Specification* (**`GPD_SPE_010`**)
  - *TEE Client API Specification* (**`GPD_SPE_007`**): defines communication interface between Rich OS apps and trusted apps.
- **OP-TEE**: open portable trusted execution environment in compliance with GlobalPlatform specs.

GlobalPlatform TEE API Specification



OP-TEE SDK Design



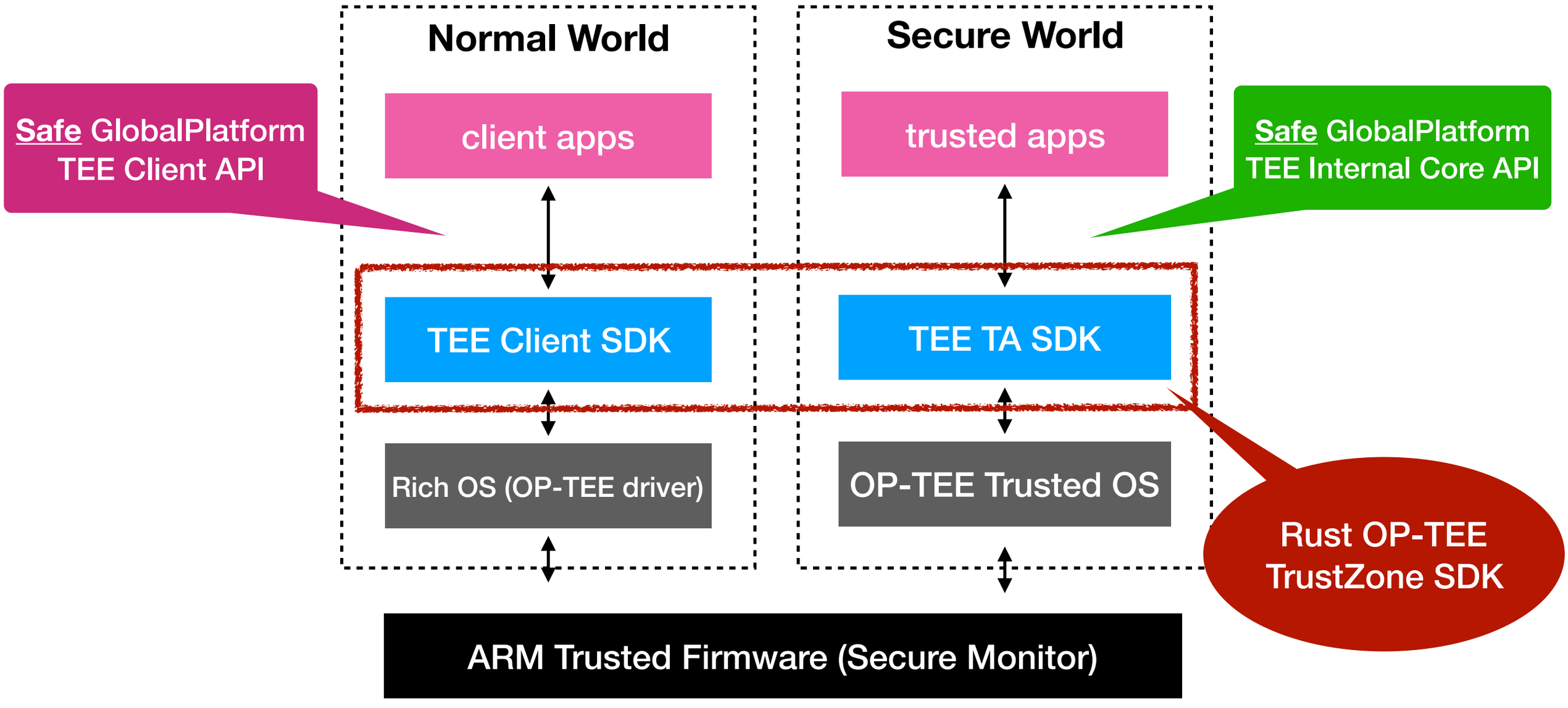
Memory-Safety

- **Memory-safety issues** break security guarantees of TrustZone.

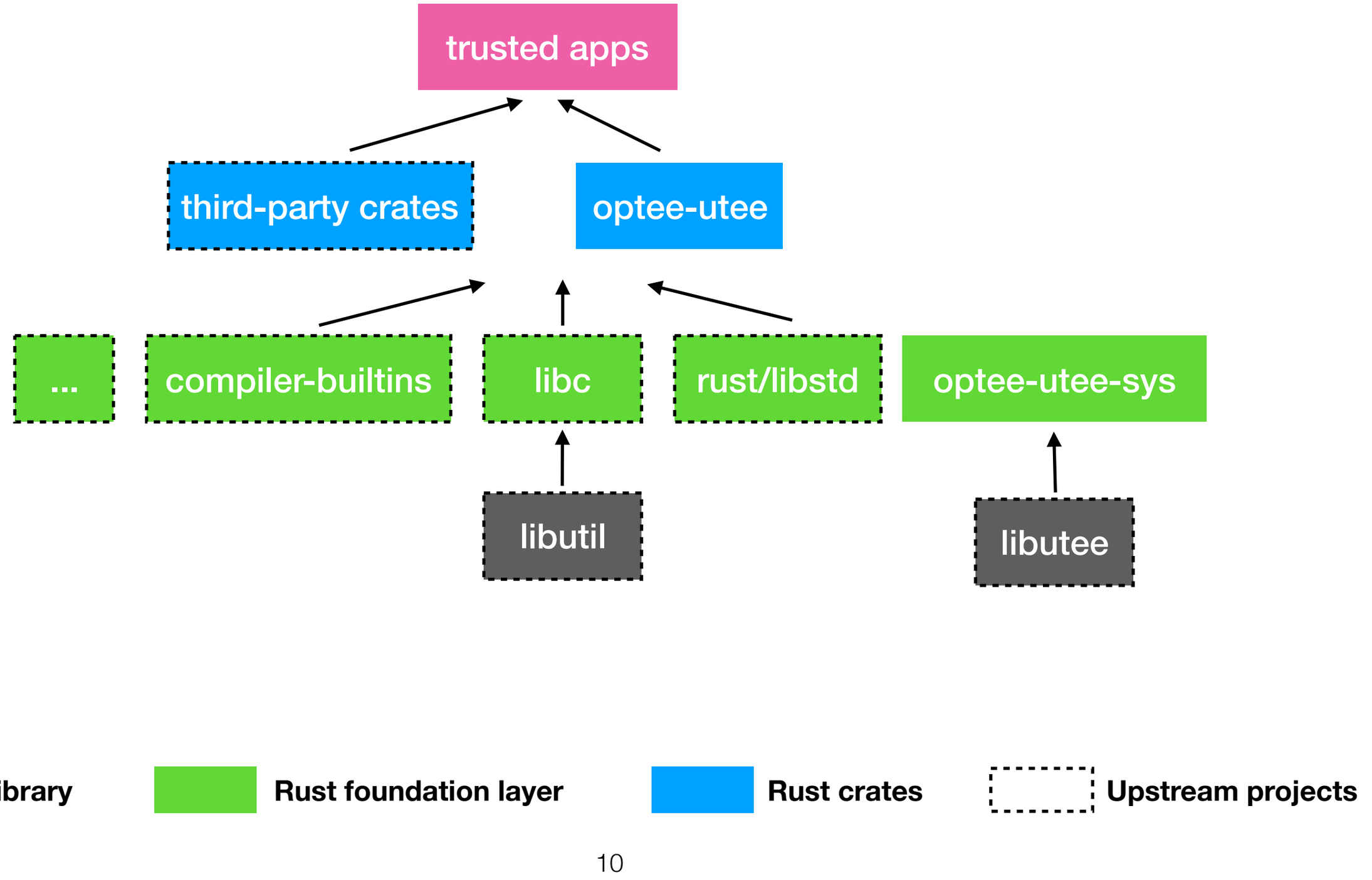
  - Qualcomm's Secure Execution Environment (QSEE) **privilege escalation** vulnerability and exploit (CVE-2015-6639) : http://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html

- Extracting Qualcomm's KeyMaster Keys - **Breaking Android Full Disk Encryption**: http://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html

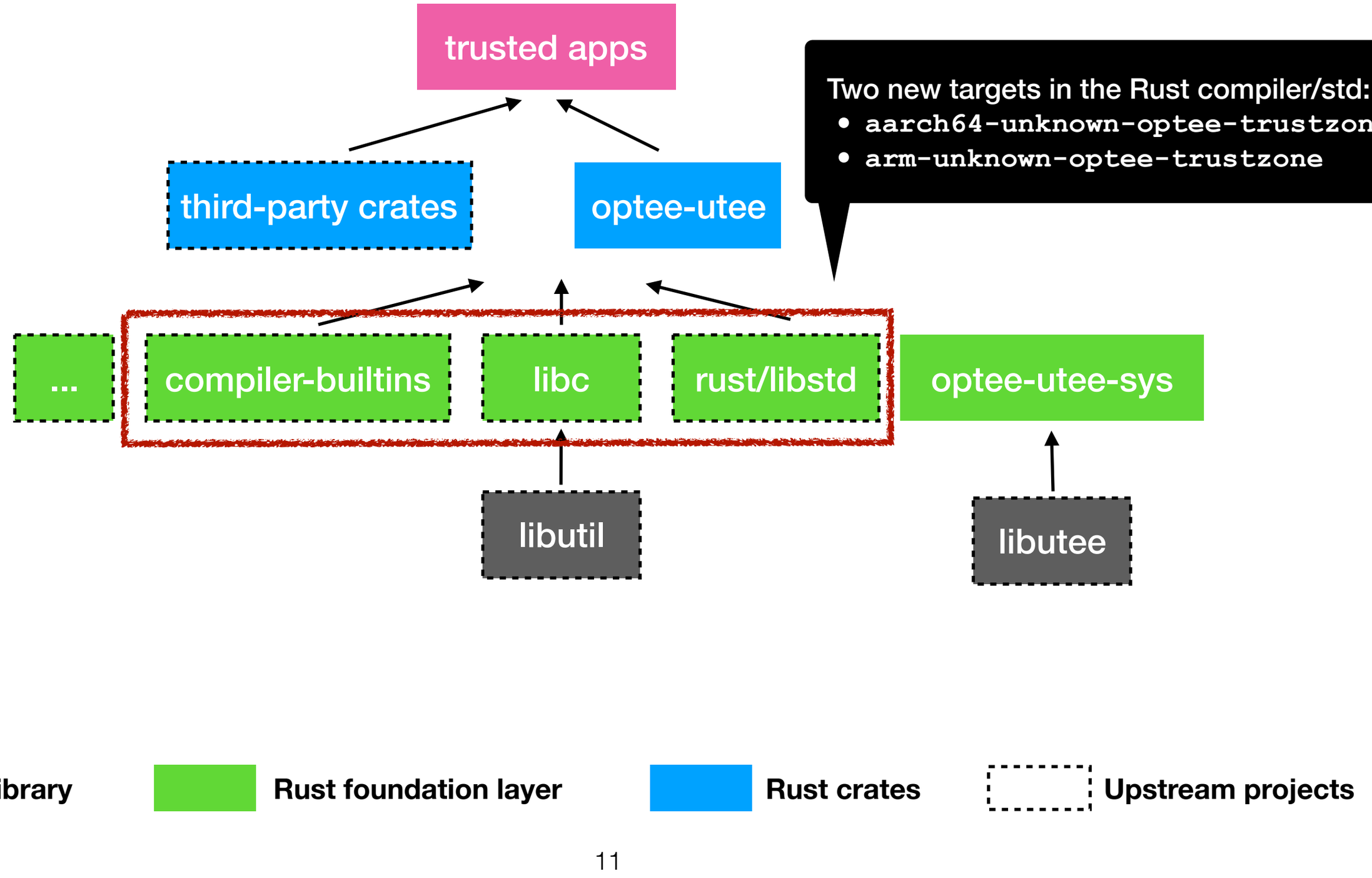
Safe SDK Design



Design of TA SDK



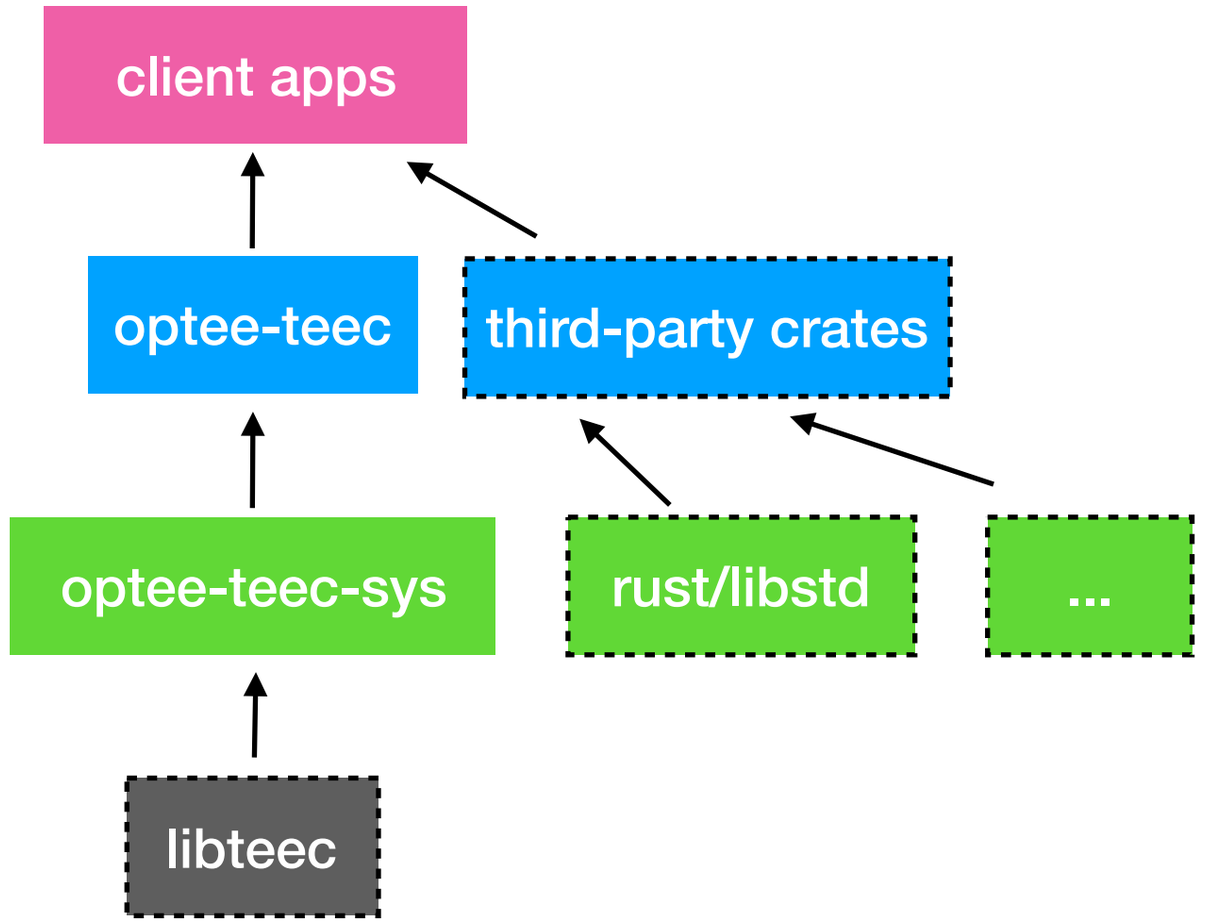
Design of TA SDK



Design of Client SDK

Client apps targets:

- `aarch64-unknown-linux-gnu`
- `arm-unknown-linux-gnu`



C library | Rust foundation layer | Rust crates | Upstream projects

Project Structure

- **Rust OP-TEE TrustZone SDK**: https://github.com/mesalock-linux/rust-optee-trustzone-sdk

- **Rust**: https://github.com/mesalock-linux/rust

- **Rust `libc`**: https://github.com/mesalock-linux/libc.git

- **Rust `compiler-builtins`**: https://github.com/mesalock-linux/compiler-builtins.git

- **Wiki**: https://github.com/mesalock-linux/rust-optee-trustzone-sdk/wiki

Project Structure

- **`optee-teec`**: client-side Rust library (LoC: ~933)
- **`optee-utee`**: TA-side Rust library (LoC: ~2827)
- **`optee`**: upstream optee library (**`optee_client`**, **`optee_os`**)
- **`rust`**: modified Rust including
  - **`rust:`** ~29 files changed, 1800 insertions
  - **`libc:`** ~4 files changed, 131 insertions
  - **`compiler-builtins:`** ~3 files changed, 3 insertions(+), 1 deletion(-)
- **`examples`**: **`hello_world`**, **`aes`**, **`hotp`**, **`random`**, **`secure_storage`**, and **`serde`** (LoC: ~3373)

Project Structure - rust/libstd

```
src/librustc_target/spec/aarch64_unknown_optee_trustzone.rs
```

```
src/libstd/sys/optee/alloc.rs
src/libstd/sys/optee/args.rs
src/libstd/sys/optee/backtrace.rs
src/libstd/sys/optee/cmath.rs
src/libstd/sys/optee/condvar.rs
src/libstd/sys/optee/env.rs
src/libstd/sys/optee/fs.rs
src/libstd/sys/optee/io.rs
src/libstd/sys/optee/memchr.rs
src/libstd/sys/optee/mod.rs
src/libstd/sys/optee/mutex.rs
```

```
src/libstd/sys/optee/net.rs
src/libstd/sys/optee/os.rs
src/libstd/sys/optee/os_str.rs
src/libstd/sys/optee/path.rs
src/libstd/sys/optee/pipe.rs
src/libstd/sys/optee/process.rs
src/libstd/sys/optee/rwlock.rs
src/libstd/sys/optee/stack_overflow.rs
src/libstd/sys/optee/stdio.rs
src/libstd/sys/optee/thread.rs
src/libstd/sys/optee/thread_local.rs
src/libstd/sys/optee/time.rs
```

Example: alloc.rs

```rust
use crate::libc;
use crate::alloc::{GlobalAlloc, Layout, System};

#[stable(feature = "alloc_system_type", since = "1.28.0")]
unsafe impl GlobalAlloc for System {
    #[inline]
    unsafe fn alloc(&self, layout: Layout) -> *mut u8 {
        libc::malloc(layout.size()) as *mut u8
    }

    #[inline]
    unsafe fn alloc_zeroed(&self, layout: Layout) -> *mut u8 {
        libc::calloc(layout.size(), 1) as *mut u8
    }

    #[inline]
    unsafe fn dealloc(&self, ptr: *mut u8, _layout: Layout) {
        libc::free(ptr as *mut libc::c_void)
    }

    #[inline]
    unsafe fn realloc(&self, ptr: *mut u8, _layout: Layout, new_size: usize) -> *mut u8 {
        libc::realloc(ptr as *mut libc::c_void, new_size) as *mut u8
    }
}
```

The underlying library of libc is `libutil` from OP-TEE

Example: thread.rs

```rust
use crate::boxed::FnBox;
use crate::ffi::CStr;
use crate::io;
use crate::sys::{unsupported, Void};
use crate::time::Duration;

pub struct Thread(Void);

pub const DEFAULT_MIN_STACK_SIZE: usize = 4096;

impl Thread {
    // unsafe: see thread::Builder::spawn_unchecked for safety requirements
    pub unsafe fn new(_stack: usize, _p: Box<dyn FnBox()>)
        -> io::Result<Thread>
    {
        unsupported()
    }

    pub fn yield_now() {
        panic!("unsupported")
    }

    pub fn set_name(_name: &CStr) {
        panic!("unsupported")
    }

    pub fn sleep(_dur: Duration) {
        panic!("unsupported");
    }

    pub fn join(self) {
        match self.0 {}
    }
}

pub mod guard {
    pub type Guard = !;
    pub unsafe fn current() -> Option<Guard> { None }
    pub unsafe fn init() -> Option<Guard> { None }
}
```

Thread is not supported in OP-TEE OS. Currently, we will raise a panic.

Getting Started with QEMU

- Rust OP-TEE TrustZone: https://github.com/mesalock-linux/rust-optee-trustzone-sdk/blob/master/README.md

- QEMU for ARMv8: https://github.com/mesalock-linux/rust-optee-trustzone-sdk/wiki/Getting-started-with-OPTEE-for-QEMU-ARMv8

Getting Started with QEMU

- Clone the project and initialize related submodules

```
$ git clone git@github.com:mesalock-linux/rust-optee-
trustzone-sdk.git
$ cd rust-optee-trustzone-sdk
$ git submodule update --init
$ (cd rust/compiler-builtins && git submodule update --init
libm compiler-rt)
$ (cd rust/rust && git submodule update --init src/stdsimd)
```

Getting Started with QEMU

- Clone the project and initialize related submodules
- Install dependencies
- Use Docker instead

```
$ docker build -t rust-optee-trustzone-sdk - < Dockerfile
```

```
$ docker run --rm -it -v$(pwd):/rust-optee-trustzone-sdk \
  -w /rust-optee-trustzone-sdk rust-optee-trustzone-sdk
```

Build Examples

```
$ make optee
$ source environment
$ make examples
```

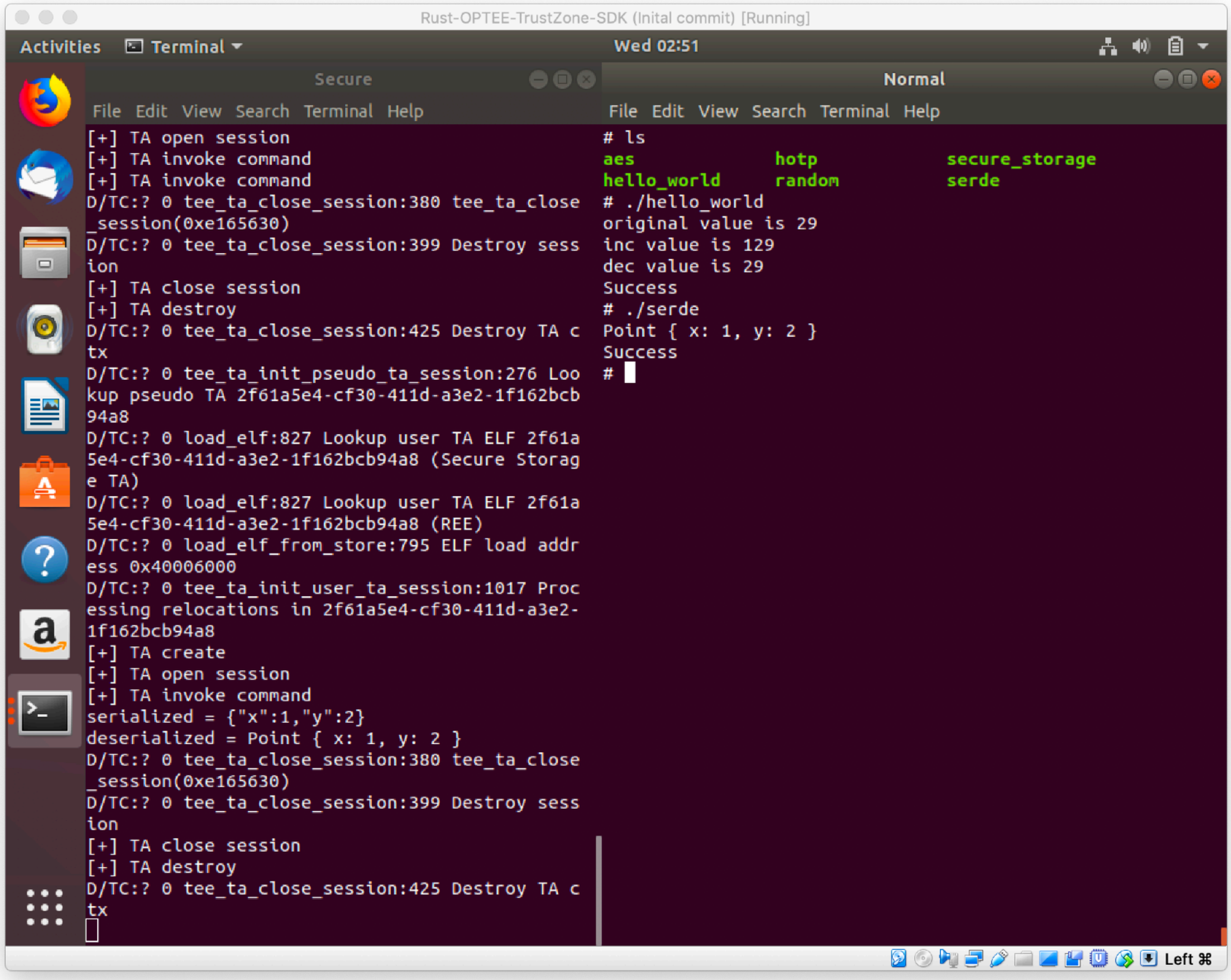
Getting Started with QEMU

- Download OP-TEE for QEMU ARMv8 source code.
- Build OP-TEE for QEMU ARMv8 and images.

```
$ docker run --rm -it -v$(pwd):/rust-optee-trustzone-sdk \
    -w /rust-optee-trustzone-sdk \
    mesalocklinux/rust-optee-trustzone-sdk-qemuv8-ci \
    bash -c "cd ci && ./ci_hello_world.sh"
```

- Login (username: root) and Mount shared folder in QEMU guest system.
- Copy TAs to corresponding directory.

Example - Demo in QEMU



Example - Client (Initial Design)

92 lines (82 sloc) | 2.5 KB — Raw | Blame | History

```rust
#![allow(non_upper_case_globals)]
#![allow(non_camel_case_types)]
#![allow(non_snake_case)]

extern crate optee_teec;
pub use libc::*;
pub use optee_teec::*;
use std::ptr;

pub const TA_HELLO_WORLD_CMD_INC_VALUE: u32 = 0;
pub const TA_HELLO_WORLD_CMD_DEC_VALUE: u32 = 1;

pub fn main() {
    let mut res: TEEC_Result;
    let mut ctx: TEEC_Context = TEEC_Context {
        fd: 0,
        reg_mem: true,
    };
    let mut sess: TEEC_Session = TEEC_Session {
        ctx: &mut ctx,
        session_id: 0,
    };

    let param1: TEEC_Parameter = TEEC_Parameter {
        value: TEEC_Value { a: 0, b: 0 },
    };
    let param2: TEEC_Parameter = TEEC_Parameter {
        value: TEEC_Value { a: 0, b: 0 },
    };
    let param3: TEEC_Parameter = TEEC_Parameter {
        value: TEEC_Value { a: 0, b: 0 },
    };
    let param4: TEEC_Parameter = TEEC_Parameter {
        value: TEEC_Value { a: 0, b: 0 },
    };
    let param_g: [TEEC_Parameter; 4] = [param1, param2, param3, param4];

    let mut op = TEEC_Operation {
        started: 0,
        param_types: 0,
        params: param_g,
        session: &mut sess,
    };
    let mut err_origin: uint32_t = 0;
    let mut uuid = TEEC_UUID {
        time_low: 0x0abcf200,
        time_mid: 0x2450,
        time_hi_and_version: 0x11e4,
        clock_seq_and_node: [0xab, 0xe2, 0x00, 0x02, 0xa5, 0xd5, 0xc5, 0x1b],
    };

    unsafe {
        res = TEEC_InitializeContext(ptr::null_mut() as *mut c_char, &mut ctx);
        if res != TEEC_SUCCESS {
            println!("Init error.\0");
            return;
        }

        res = TEEC_OpenSession(
            &mut ctx,
            &mut sess,
            &mut uuid,
            TEEC_LOGIN_PUBLIC,
            ptr::null() as *const c_void,
            ptr::null_mut() as *mut TEEC_Operation,
            &mut err_origin,
        );
        if res != TEEC_SUCCESS {
            println!("Open session error.\0");
            return;
        }

        op.param_types = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT, TEEC_NONE, TEEC_NONE, TEEC_NONE);
        op.params[0].value.a = 29;
        println!("original value is {}", op.params[0].value.a);
        res = TEEC_InvokeCommand(
            &mut sess,
            TA_HELLO_WORLD_CMD_INC_VALUE,
            &mut op,
            &mut err_origin,
        );
        if res != TEEC_SUCCESS {
            println!("Execute command error.\0");
            return;
        }
        println!("update value is {}", op.params[0].value.a);

        TEEC_CloseSession(&mut sess);
```

```
raw::TEEC_Context
raw::TEEC_Session
raw::TEEC_Parameter
raw::TEEC_Operation
raw::TEEC_InitializeContext
raw::TEEC_OpenSession
raw::TEEC_InvokeCommand
raw::TEEC_CloseSession
raw::TEEC_FinalizeContext
```

```
unsafe {


}
```

Example - Client (Current Design)

```rust
use optee_teec::{Context, Operation, ParamType, Session, Uuid};
use optee_teec::{ParamNone, ParamValue};
use proto::{self, Command};

fn hello_world(session: &mut Session) -> optee_teec::Result<()> {
    let p0 = ParamValue::new(29, 0, ParamType::ValueInout);
    let mut operation = Operation::new(0, p0, ParamNone, ParamNone, ParamNone);

    println!("original value is {:?}", operation.parameters().0.a());

    session.invoke_command(Command::IncValue as u32, &mut operation)?;
    println!("inc value is {:?}", operation.parameters().0.a());

    session.invoke_command(Command::DecValue as u32, &mut operation)?;
    println!("dec value is {:?}", operation.parameters().0.a());
    Ok(())
}

fn main() -> optee_teec::Result<()> {
    let mut ctx = Context::new()?;
    let uuid = Uuid::parse_str(proto::UUID).unwrap();
    let mut session = ctx.open_session(uuid)?;

    hello_world(&mut session)?;

    println!("Success");
    Ok(())
}
```

`ParamValue::new()`

`Operation::new()`

`session.invoke_command()`

`Context::new()`

`ctx.open_session()`

Example - Trusted App (First Commit)

```
#![allow(non_upper_case_globals)]
#![allow(non_camel_case_types)]
#![allow(non_snake_case)]

extern crate optee_utee;
pub use optee_utee::*;

#[no_mangle]
pub extern "C" fn TA_CreateEntryPoint() -> TEE_Result {
    return TEE_SUCCESS;
}

#[no_mangle]
pub extern "C" fn TA_DestroyEntryPoint() {
}

#[no_mangle]
pub extern "C" fn TA_OpenSessionEntryPoint(_paramTypes: ParamTypes, _params: TEE_Param, _sessionContext: SessionP) -> TEE_Resul
    return TEE_SUCCESS;
}

#[no_mangle]
pub extern "C" fn TA_CloseSessionEntryPoint(_sessionContext: SessionP) {
}

#[no_mangle]
pub extern "C" fn TA_InvokeCommandEntryPoint(_sessionContext: SessionP, _c
    match _commandID {
        0 => {
            unsafe { _params[0].value.a += 121; }
        },
        1 => {
            unsafe { _params[0].value.a -= 21; }
        },
        _ => {
            return TEE_ERROR_BAD_PARAMETERS;
        }
    }
    return TEE_SUCCESS;
}
```

```
#[no_mangle]

pub extern "C" fn TA_CreateEntryPoint() -> TEE_Result {
    return TEE_SUCCESS;
}
```

```
#[no_mangle]

pub extern "C" fn TA_OpenSessionEntryPoint(
    _paramTypes: ParamTypes,
    _params: TEE_Param,
    _sessionContext: SessionP) -> TEE_Result {
    return TEE_SUCCESS;
}
```

```
0 => {

    unsafe { _params[0].value.a += 121; }

},
```

Example - Trusted App (Current Design)

```rust

#[ta_create]
fn create() -> Result<()> {
    trace_println!("[+] TA create");
    Ok(())
}

#[ta_open_session]
fn open_session(_params: &mut Parameters) -> Result<()> {
    trace_println!("[+] TA open session");
    Ok(())
}

#[ta_close_session]
fn close_session() {
    trace_println!("[+] TA close session");
}

#[ta_destroy]
fn destroy() {
    trace_println!("[+] TA destroy");
}

#[ta_invoke_command]
fn invoke_command(cmd_id: u32, params: &mut Parameters) -> Result<()> {
    trace_println!("[+] TA invoke command");
    let mut values = unsafe { params.0.as_value().unwrap() };
    match Command::from(cmd_id) {
        Command::IncValue => {
            values.set_a(values.a() + 100);
            Ok(())
        }
        Command::DecValue => {
            values.set_a(values.a() - 100);
            Ok(())
        }
        _ => Err(Error::new(ErrorKind::BadParameters)),
    }
}
```

#[ta_create]

#[ta_open_session]

#[ta_close_session]

#[ta_destory]

#[ta_invoke_command]

Example - Trusted App (Current Design)

```
48  // TA configurations
49  const TA_FLAGS: u32 = 0;
50  const TA_DATA_SIZE: u32 = 32 * 1024;
51  const TA_STACK_SIZE: u32 = 2 * 1024;
52  const TA_VERSION: &[u8] = b"0.1\0";
53  const TA_DESCRIPTION: &[u8] = b"This is a hello world ex
54  const EXT_PROP_VALUE_1: &[u8] = b"Hello World TA\0";
55  const EXT_PROP_VALUE_2: u32 = 0x0010;
56  const TRACE_LEVEL: i32 = 4;
57  const TRACE_EXT_PREFIX: &[u8] = b"TA\0";
58  const TA_FRAMEWORK_STACK_SIZE: u32 = 2048;
59
60  include!(concat!(env!("OUT_DIR"), "/user_ta_header.rs"));
```

TA configurations

TA_DATA_SIZE: heap size
TA_STACK_SIZE: stack size

Include some static data structures

Example - Project Structure

- **`host/`**: source code of the client app

  `arm-unknown-linux-gnu`
  `aarch64-unknown-linux-gnu`

- **`ta/`**: source code of TA

  `arm-unknown-optee-trustzone`
  `aarch64-unknown-optee-trustzone`

  - **`ta.lds`**: linker script
  - **`Xargo.toml`**: "`Cargo.toml`" for cross compilation
  - **`ta_static.rs`**: some static data structure for TA
- **`proto/`**: shared data structure and configurations like a protocol
- **`Makefile`**: Makefile to build host and client
- **`uuid.txt`**: UUID for TA, randomly generated if the file does not exist.

Example - Use Serde

```
[package]
name = "ta"
version = "0.1.0"
authors = ["The Rust OP-TEE TrustZone SDK Project Developers"]
license = "Apache-2.0"
repository = "https://github.com/mesalock-linux/rust-optee-trustzone-sdk.git"
description = "An example of Rust OP-TEE TrustZone SDK."
edition = "2018"

[dependencies]
libc = { path = "../../../rust/libc" }
optee-utee-sys = { path = "../../../optee-utee/optee-utee-sys" }
optee-utee = { path = "../../../optee-utee" }
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"

[build_dependencies]
uuid = { version = "0.7", features = ["v4"] }
```

serde / serde_json

Example - Use Serde

```
#[ta_invoke_command]
fn invoke_command(cmd_id: u32, _params: &mut Parameters) -> Result<()> {
    trace_println!("[+] TA invoke command");
    match Command::from(cmd_id) {
        Command::DefaultOp => {
            let point = Point { x: 1, y: 2 };

            // Convert the Point to a JSON string.
            let serialized = serde_json::to_string(&point).unwrap();

            // Prints serialized = {"x":1,"y":2}
            trace_println!("serialized = {}", serialized);

            // Convert the JSON string back to a Point.
            let deserialized: Point = serde_json::from_str(&serialized).unwrap();

            // Prints deserialized = Point { x: 1, y: 2 }
            trace_println!("deserialized = {:?}", deserialized);

            Ok(())
        }
        _ => Err(Error::new(ErrorKind::BadParameters)),
    }
}
```

Use **`serde`** to handle invoke command

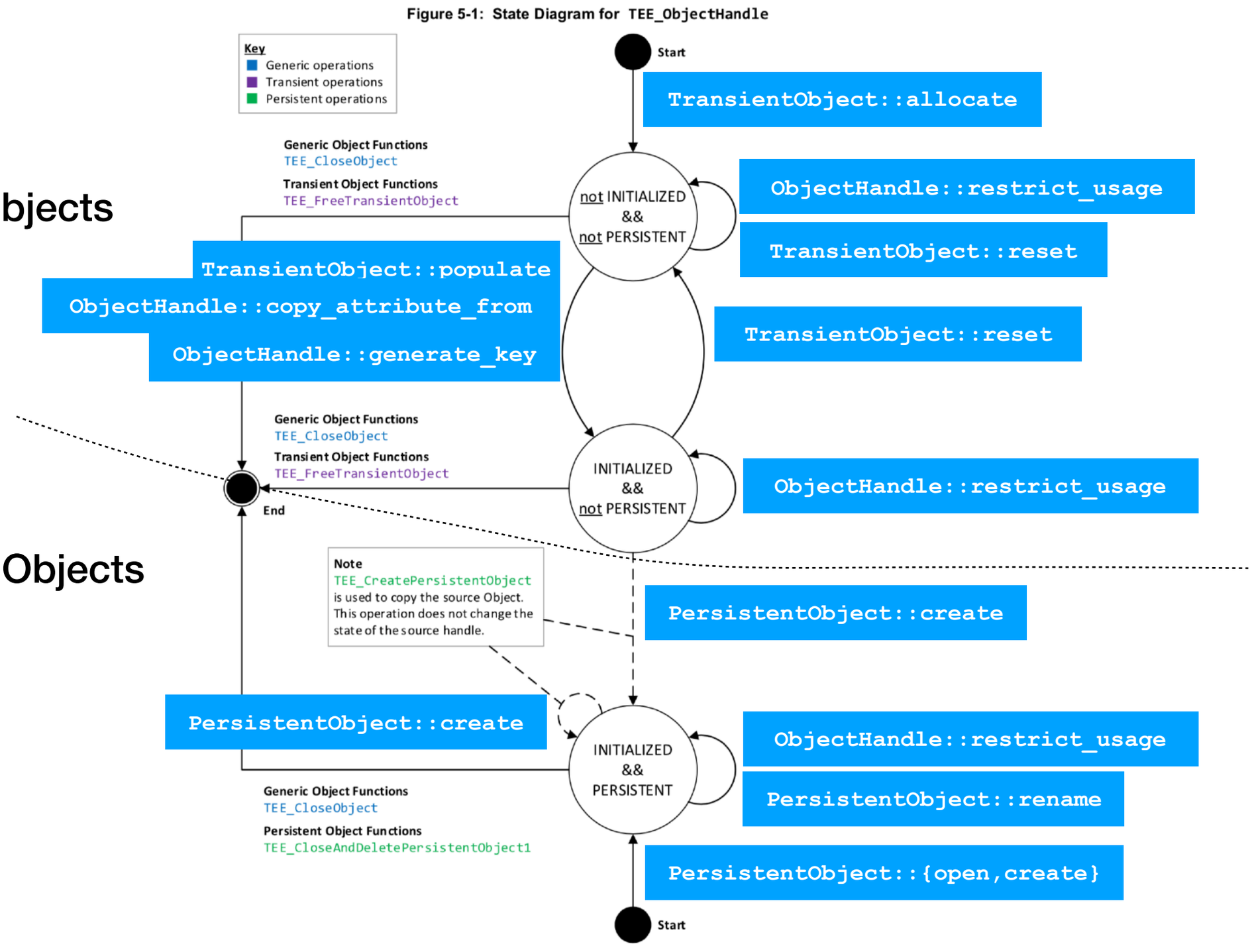
Trusted Storage API for Data and Keys

- Trusted Storage Spaces
- Transient objects
- Persistent objects

Trusted Storage API for Data and Keys

Transient Objects

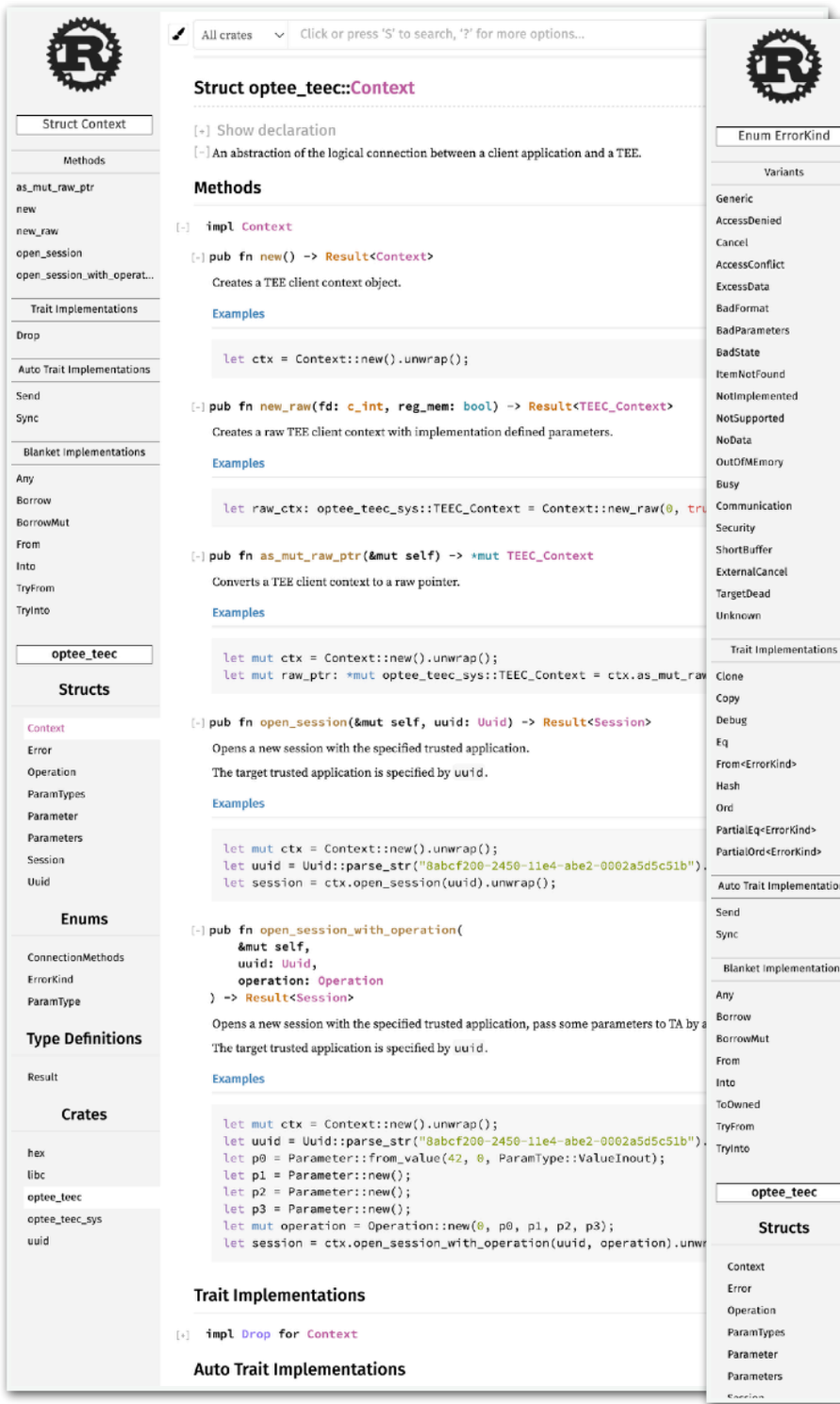
Persistent Objects

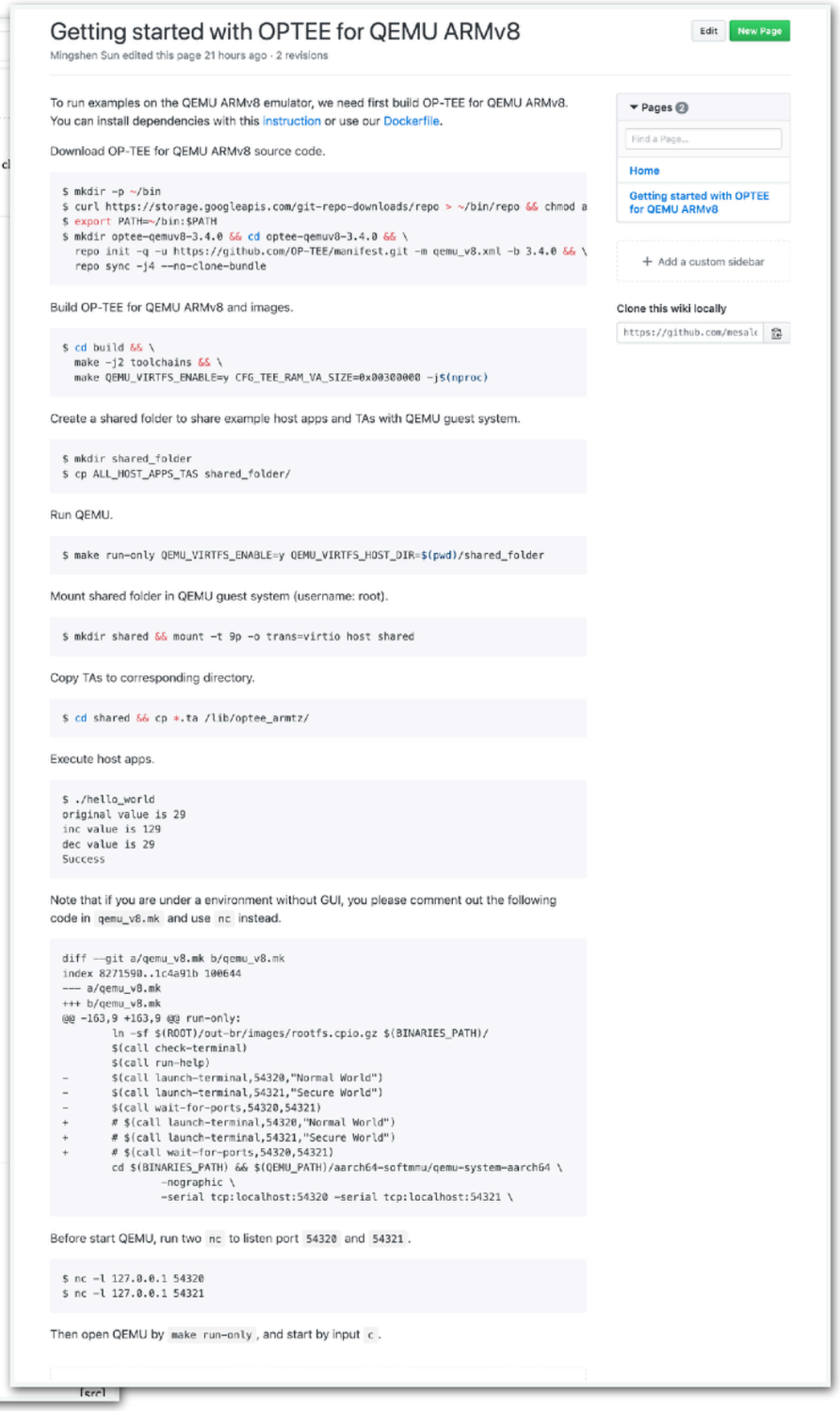


Other Examples

- **`hello_world`**: minimal project structure
- **`aes`**: crypto, shared memory APIs
- **`hotp`**: crypto APIs
- **`random`**: crypto APIs
- **`secure_storage`**: secure object related APIs
- **`serde`**: Rust third-party crates for de/serialization

Documents & Wiki





Test

- **ctest**: Automated testing of FFI bindings in Rust. This repository is intended to validate the `*-sys` crates that can be found on crates.io to ensure that the APIs in Rust match the APIs defined in C.

- QEMU integration tests, no unit tests for now

- Travis CI

Run/Test Examples in QEMU

```
$ docker run --rm -it \
    -v$(pwd):/rust-optee-trustzone-sdk \
    -w /rust-optee-trustzone-sdk \
    mesalocklinux/rust-optee-trustzone-sdk-qemuv8-ci \
    bash -c "cd ci && ./ci_hello_world.sh"
```

Roadmap

- **April**: open source
- **May**: trusted storage API design, cryptographic operations API design, TEE arithmetical API design, and more third-party Rust crates
- **Jun**: push modified Rust compiler/std to upstream and make OP-TEE TrustZone as an official target.
- **2019 Q3/4**: more trusted apps such as secure key service, remote attestation, fTPM, and machine learning algorithm.

Summary

- TrustZone 为手机、嵌入式设备提供安全的可信执行环境，用于包括安全支付、密钥管理、模型保护等场景。但是由于内存安全问题，TrustZone 中运行的安全应用 (trusted application, TA) 的安全性大打折扣。

- Rust OP-TEE TrustZone SDK 为当今广泛使用的开源 TrustZone 实现 OP-TEE 提供了一套**内存安全**、**使用方便**的 SDK。SDK 基于 GlobalPlatform 的 TEE 标准，为开发者提供标准的开发接口。除此之外 Rust OP-TEE TrustZone SDK **支持标准库**和**第三方库**，提高了 TA 的开发速度，并扩展了 TrustZone 的应用场景。同时，它是 **MesaTEE 项目**的一部分。

- **License**: Apache v2