Independent subsidiary of Qualcomm Innovation Center, Inc.

Rusty OP-TEE Trusted Applications

Sumit Garg

Senior Staff Engineer, QUALCOMM India Private Limited

#OSSummit

Who am I?

- A member of the Foundries.io team. Foundries.io is an independent subsidiary of Qualcomm Innovation Center, Inc.

- Have a keen interest in platform security
  - Linux kernel: TEE subsystem reviewer
  - OP-TEE: Author for Rust no-std Trusted Applications support
  - TF-A and OP-TEE: firmware encryption framework maintainer
  - TF-A and OP-TEE: platform maintainer

- Other areas of interest
  - U-Boot: OF_UPSTREAM devicetree maintainer
  - ftrace for OP-TEE
  - meta-arm-toolchain

Open Portable TEE (OP-TEE)

An open source TEE implementation with **mainline** Linux kernel driver

Supported architectures: arm, arm64 and risc-v (early stages)

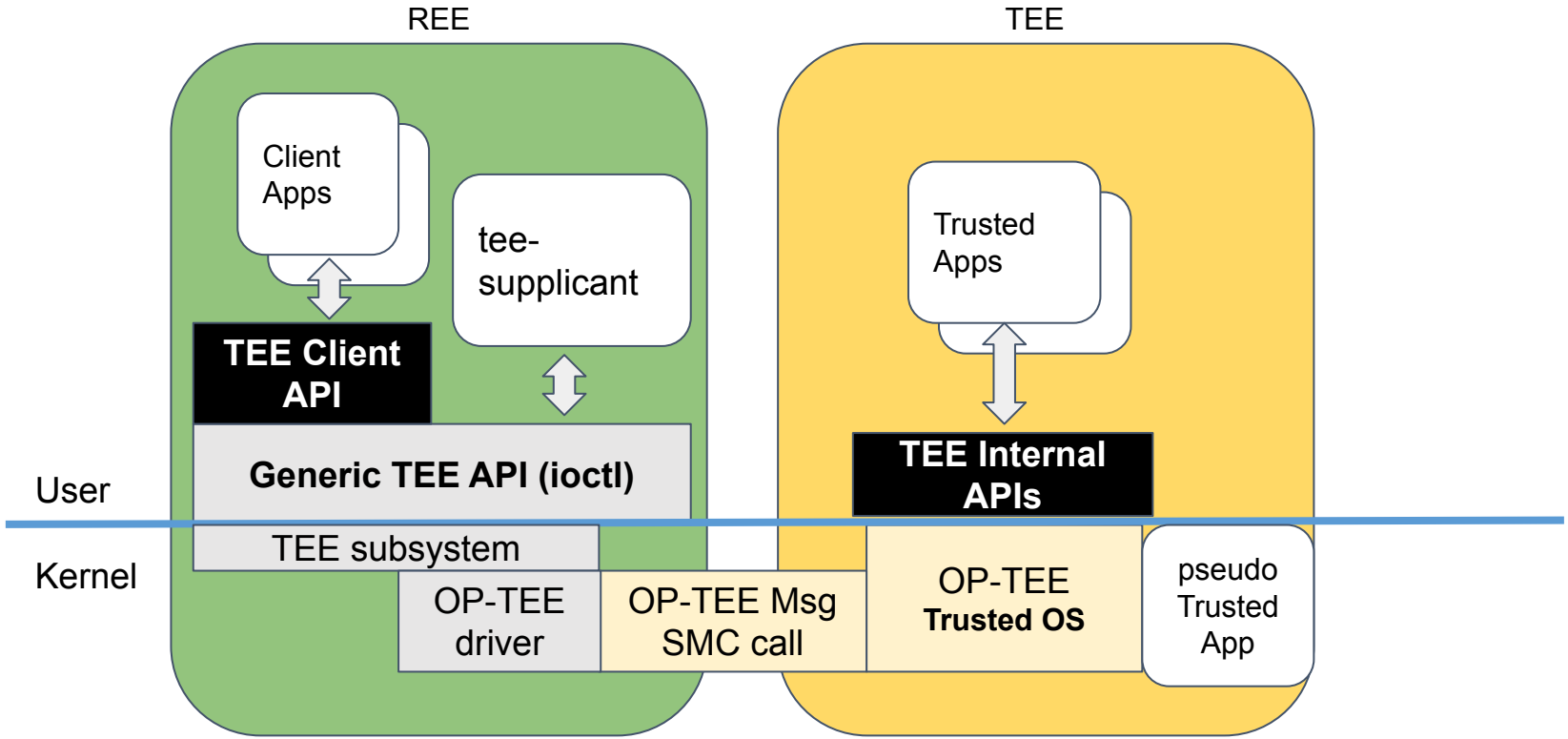
Design goals:

- Isolation
- Small footprint
- Portability

You don’t need to have actual hardware to get hands-on experience with OP-TEE:

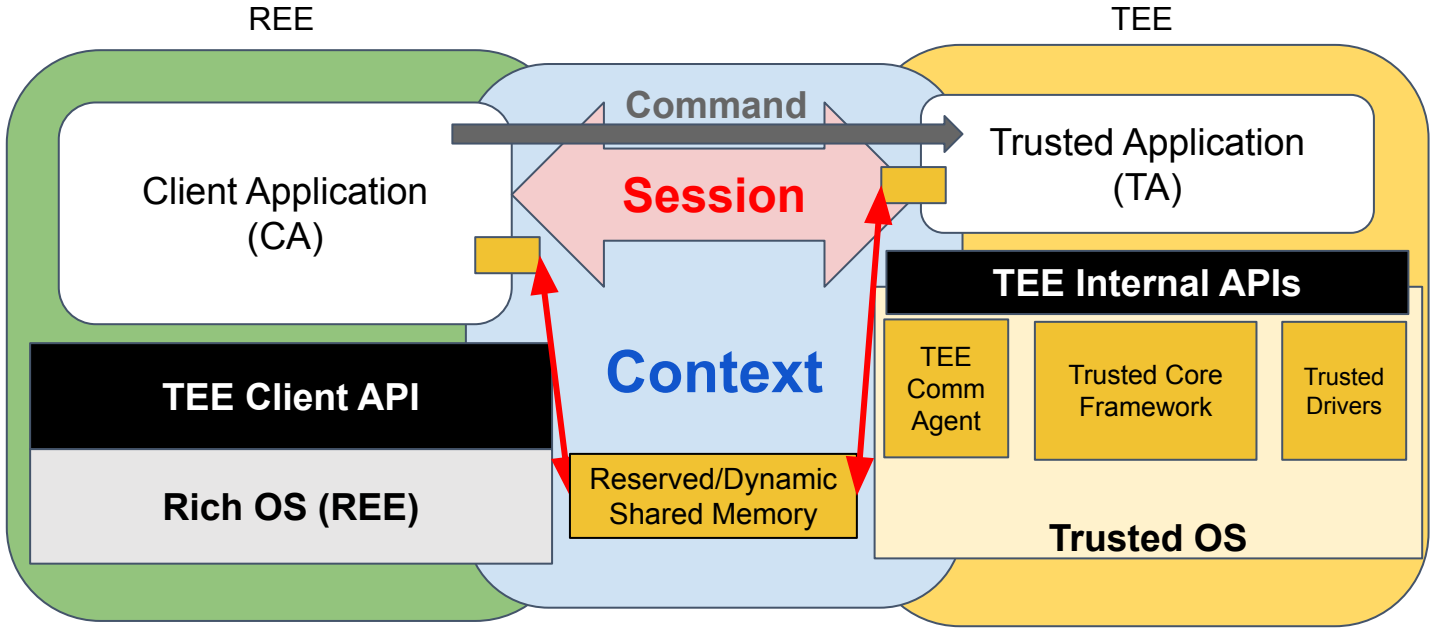
- Use Qemu, steps here.

OP-TEE architecture



Foundries.io

OP-TEE communication stack



TEEC_{Initialize/Finalize}Context()

TEEC_{Open/Close}Session()

TEEC_InvokeCommand()

TEEC_{Allocate/Register/Free}SharedMemory()

Foundries.io

Why rusting OP-TEE?

Memory safety is critical to OP-TEE because its applications provide secure interfaces that can be used by the (potentially compromised) rich OS to manipulate user's/system credentials, keys and confidential data.

In particular Rust language memory-safety guarantees can be a huge step forward to harden the OP-TEE environment against any memory based exploits.

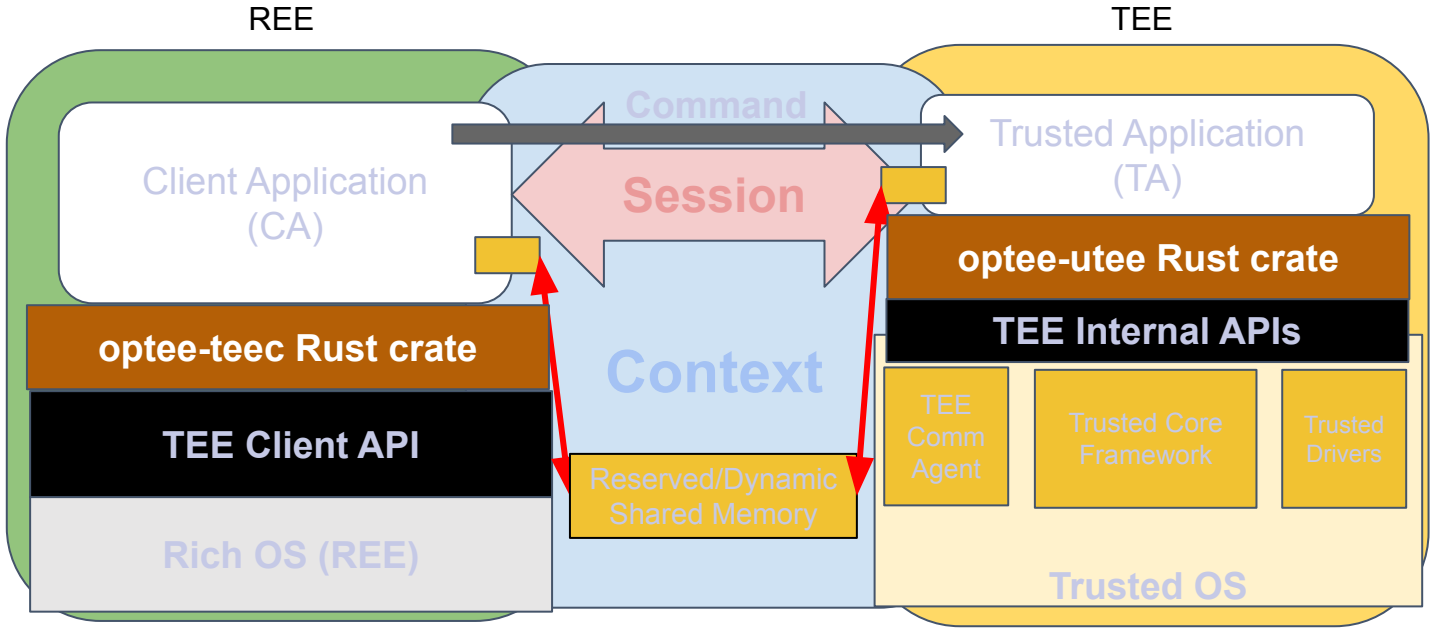
Rusty OP-TEE story: Inception

Thanks to efforts from Teaclave TrustZone SDK project team to bring Rust support to OP-TEE. The initial target for this SDK is to support Trusted Applications (TAs) written in Rust.

Initial Rust SDK support for TAs:

- Custom Rust toolchain OP-TEE target to support Rust std libraries
  - aarch64-unknown-optee
  - arm-unknown-optee
- Bind to a particular Rust toolchain version
- Bulkier and less performant (approx 35% less) compared to C counterparts

OP-TEE communication stack (Rust)




```
let mut ctx = optee_teec::Context::new()?;
let mut op = optee_teec::Operation::new(<p>)?;
```

```
let mut session = ctx.open_session(uuid)?;
session.invoke_command(id, &mut op)?;
```


Foundries.io

Rusty OP-TEE story: why no-std?

OP-TEE major design goal: Small footprint

OP-TEE user-space environment for Trusted Applications (TAs) is constrained:

- Limited TA memory resources (upto few megabytes)
- No user-space threading support
- Limited networking support (GP TEE sockets interface)
- No direct filesystem storage
- Limited secure storage (GP TEE secure storage interface)

Rust no-std environment perfectly fits these requirements, allows usage of upstream Rust tier-1 Arm64 toolchain target.

Rusty OP-TEE story: no-std

Added support for rust no-std environment for OP-TEE Trusted Applications

Benefits of Rust no-std TAs:

- First class citizen using Rust Tier 1 Arm64 toolchain
- Performance parity with C counterpart TAs
- Approx. 70K - 80K binary size reduction compared to std TAs
- no-std support is gaining traction in the Rust community

However, the SDK still supports std TAs for some complex use-cases.

Tips to write no-std TAs

OP-TEE crates can be found on crates.io as:

- optee-teec: wrapper crate for TEE client API
- optee-utee: wrapper crate for TEE internal core API
- optee-utee-build: a builder crate for TAs

Tips to write no-std TAs - ||

Standalone compile optee_os and optee_client, for eg. for a Qemu machine:

```
# Install prerequisites

$ cd optee_os/
$ make PLATFORM=vexpress-qemu_armv8a CROSS_COMPILE64=aarch64-linux-gnu-
CROSS_COMPILE32=arm-linux-gnueabihf-
$ export TA_DEV_KIT_DIR=$PWD/out/arm-plat-vexpress/export-ta_arm64/

$ cd optee_client/
$ make CROSS_COMPILE=aarch64-linux-gnu- WITH_TEEACL=0
$ export OPTEE_CLIENT_EXPORT=$PWD/out/export/
```

Tips to write no-std TAs - hello-world-rs

```
$ tree
.
├── host
│   ├── Cargo.toml
│   ├── Makefile
│   └── src
│       └── main.rs
├── proto
│   ├── Cargo.toml
│   └── src
│       └── lib.rs
└── ta
    ├── build.rs
    ├── Cargo.toml
    ├── Makefile
    └── src
        └── main.rs

7 directories, 9 files
```


Foundries.io

hello-world-rs: host

“host” crate is referred to as OP-TEE client application running in Linux user-space

host/Cargo.toml

```toml
[package]
name = "hello_world-rs"
version = "0.4.0"
edition = "2024"

[dependencies]
proto = { path = "../proto" }
optee-teec = { version = "0.4.0" }

[profile.release]
lto = true
```

hello-world-rs: host

```rust
extern crate optee_teec;
extern crate proto;

use optee_teec::{Context, Operation, ParamType, Session, Uuid};
use optee_teec::{ParamNone, ParamValue};
use proto::{UUID, 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);

    session.invoke_command(Command::IncValue as u32, &mut operation)?;
    session.invoke_command(Command::DecValue as u32, &mut operation)?;
    Ok(())
}

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

    hello_world(&mut session)?;
    Ok(())
}
```

hello-world-rs: host

host/Makefile

```
NAME := hello_world-rs

TARGET ?= aarch64-unknown-linux-gnu
CROSS_COMPILE ?= aarch64-linux-gnu-
OBJCOPY := $(CROSS_COMPILE)objcopy
LINKER_CFG := target.$(TARGET).linker=\"$(CROSS_COMPILE)gcc\"

OUT_DIR := $(CURDIR)/target/$(TARGET)/release

all: host strip

host:
	@cargo build --target $(TARGET) --release --config $(LINKER_CFG)

strip: host
	@$(OBJCOPY) --strip-unneeded $(OUT_DIR)/$(NAME) $(OUT_DIR)/$(NAME)

clean:
	@cargo clean
```

hello-world-rs: proto

“proto” crate is referred to as the protocol among client and trusted applications

proto/Cargo.toml

```
[package]
name = "proto"
version = "0.4.0"
edition = "2024"

[dependencies]
num_enum = { version = "0.7.3", default-features = false }
```

hello-world-rs: proto

proto/src/lib.rs

```
#![no_std]

extern crate num_enum;
use num_enum::{FromPrimitive, IntoPrimitive};

#[derive(FromPrimitive, IntoPrimitive)]
#[repr(u32)]
pub enum Command {
    IncValue,
    DecValue,
    #[default]
    Unknown,
}

pub const UUID: &str = "133af0ca-bdab-11eb-9130-43bf7873bf67";
```

hello-world-rs: ta

“ta” crate is referred to as the OP-TEE Trusted Application

ta/Cargo.toml

```
[package]
name = "ta"
version = "0.4.0"
edition = "2024"

[dependencies]
proto = { path = "../proto" }
optee-utee-sys = { version = "0.4.0" }
optee-utee = { version = "0.4.0" }

[build-dependencies]
proto = { path = "../proto" }
optee-utee-build = { version = "0.4.0" }

[profile.release]
panic = "abort"
lto = true
opt-level = 1
```

hello-world-rs: ta

```rust
#![no_std]
#![no_main]
extern crate optee_utee;
extern crate proto;
use optee_utee::{
    ta_close_session, ta_create, ta_destroy, ta_invoke_command, ta_open_session, trace_println,
};
use optee_utee::{Error, ErrorKind, Parameters, Result};
use proto::Command;

#[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");
}
...
```

hello-world-rs: ta

```
...
#[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_destroy]
fn destroy() {
    trace_println!("[+] TA destroy");
}

include!(concat!(env!("OUT_DIR"), "/user_ta_header.rs"))
```

hello-world-rs: ta

ta/build.rs

```
extern crate proto;
extern crate optee_utee_build;

use optee_utee_build::{TaConfig, RustEdition, Error};

fn main() -> Result<(), Error> {
    let config = TaConfig::new_default_with_cargo_env(proto::UUID)?;
    optee_utee_build::build(RustEdition::Edition2024, config)
}
```

Reference: https://github.com/apache/incubator-teaclave-trustzone-sdk/blob/main/docs/writing-rust-tas-using-optee-utee-build.md

hello-world-rs: ta

```
UUID ?= 133af0ca-bdab-11eb-9130-43bf7873bf67

TARGET ?= aarch64-unknown-linux-gnu
CROSS_COMPILE ?= aarch64-linux-gnu-
OBJCOPY := $(CROSS_COMPILE)objcopy
LINKER_CFG := target.$(TARGET).linker=\"$(CROSS_COMPILE)gcc\"

TA_SIGN_KEY ?= $(TA_DEV_KIT_DIR)/keys/default_ta.pem
SIGN := $(TA_DEV_KIT_DIR)/scripts/sign_encrypt.py
OUT_DIR := $(CURDIR)/target/$(TARGET)/release

all: ta strip sign

ta:
	@cargo build --target $(TARGET) --release --config $(LINKER_CFG)

strip: ta
	@$(OBJCOPY) --strip-unneeded $(OUT_DIR)/ta $(OUT_DIR)/stripped_ta

sign: strip
	@$(SIGN) --uuid $(UUID) --key $(TA_SIGN_KEY) --in $(OUT_DIR)/stripped_ta --out
$(OUT_DIR)/$(UUID).ta
	@echo "SIGN =>  ${UUID}"

clean:
	@cargo clean
```

Build hello-world-rs

no-std Rust TAs currently depend on some Rust nightly toolchain features, so install latest Rust nightly toolchain:

```
$ rustup toolchain install nightly
$ rustup target add aarch64-unknown-linux-gnu

$ cd hello-world-rs/host
$ make

# client app: target/aarch64-unknown-linux-gnu/release/hello_world-rs

$ cd hello-world-rs/ta
$ make

# TA: target/aarch64-unknown-linux-gnu/release/133af0ca-bdab-11eb-9130-43bf7873bf67.ta
```

Future work…

- Stable Rust toolchain support for no-std TAs
- Request for Global Platforms
  - Enable portability of Rust based TAs
  - TEE client API specification for Rust developers
  - TEE internal API specification for Rust developers
- Start Rusting OP-TEE core components
  - User-space libraries
  - pseudo TAs part of core can be a good starting point

Foundries.io

Independent subsidiary of Qualcomm Innovation Center, Inc.

Thank you

Follow us on