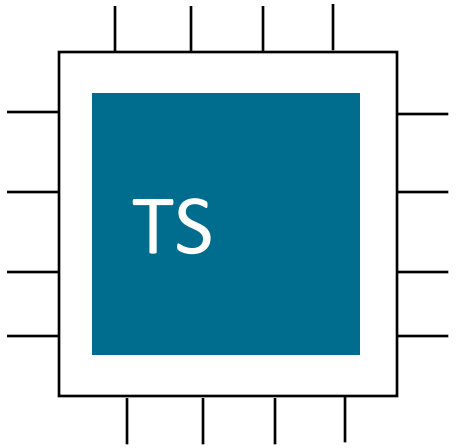
Introduction to the Trusted Services Project

Julian Hall

8 December 2020

A new *trustedfirmware.org* project

- Trusted Services (TS) is a new project under *trustedfirmware.org*.
- First published to TrustedFirmware git on 26th November 2020.
- Originates from work by the Arm OSS OP-TEE firmware team to implement PSA RoT services that can run in Secure Partitions.
- Complements existing *trustedfirmware.org* projects.



arm

What are Trusted Services?

- A general term for a firmware service that performs security related operations on behalf of clients.
- A trusted service provider runs within a secure processing environment to protect security sensitive assets from malicious software running outside of the environment.
- On Arm Cortex-A SoCs, a range of secure processing environments are available:
  - Secure partitions – managed by an SPM, implemented by:
    - A TEE such as OP-TEE
    - A dedicated SEL2 component such as Hafnium
    - As part of EL3 firmware
  - Trusted applications – managed by a TEE
  - Secure enclave – a secondary MCU
- Example services:
  - Crypto – provides cryptographic operations with a protected key store
  - Secure storage – provides protected persistent storage
  - fTPM – TPM 2.0 firmware, running as a trusted service
  - UEFI Keystore – a protected persistent store for UEFI keys

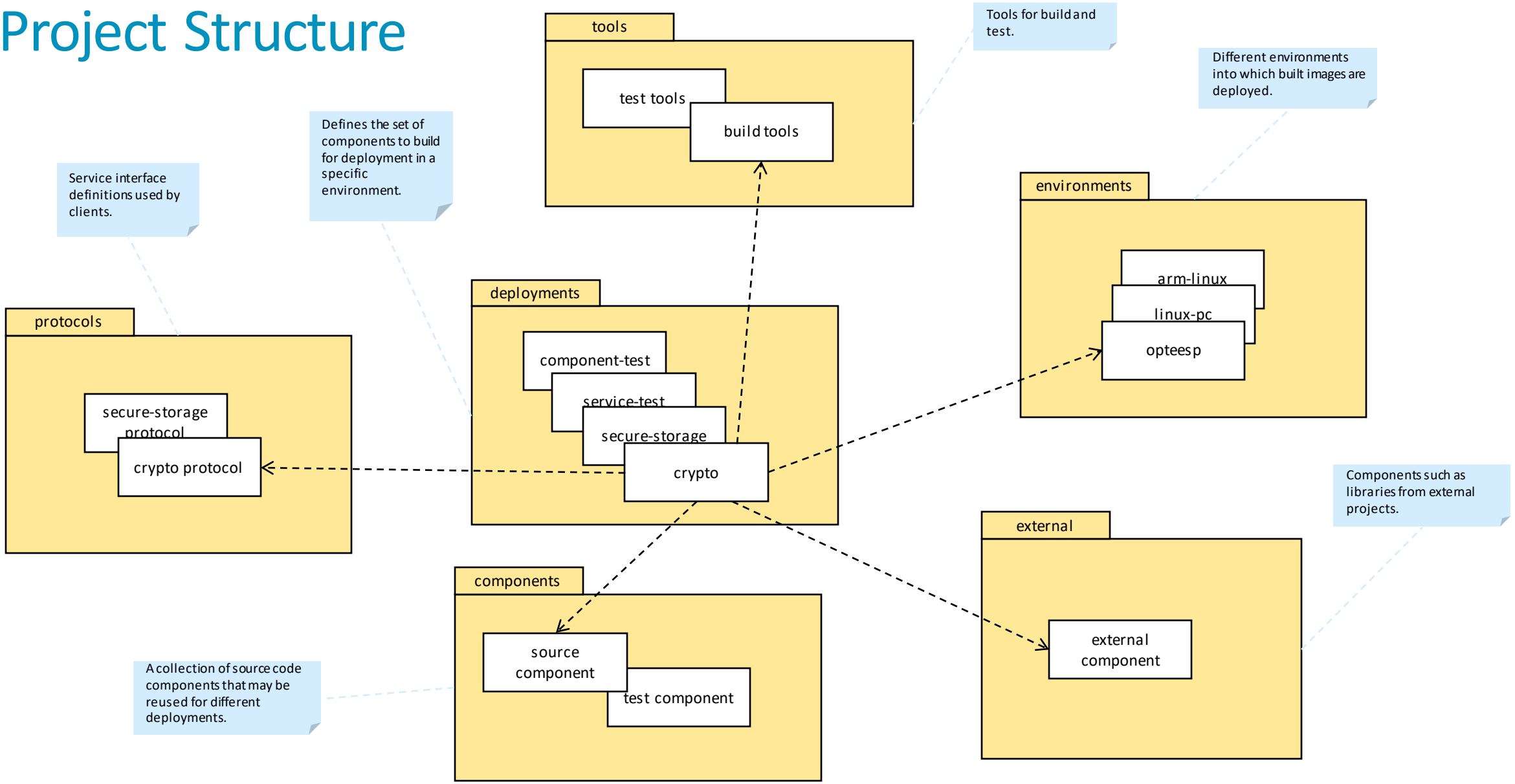
Why have a separate Trusted Services project?

- The Trusted Services project provides a home for service related components that may be integrated and deployed in different processing environments.
- The project is independent of any particular secure processing environment project.
  - E.g., Overloading the OP-TEE project with trusted service components would undermine opportunities for reuse outside of OP-TEE.
- A centralized project creates opportunities for:
  - Adopting a common framework with standard conventions and solutions.
  - Component and test-case reuse.
  - Publishing standard public interfaces.
  - Sharing security enhancements.
  - Having a common solution for build, testing and exporting to client projects.
- Opens the door for trusted service deployment on any Arm Cortex-A based device using common core components.

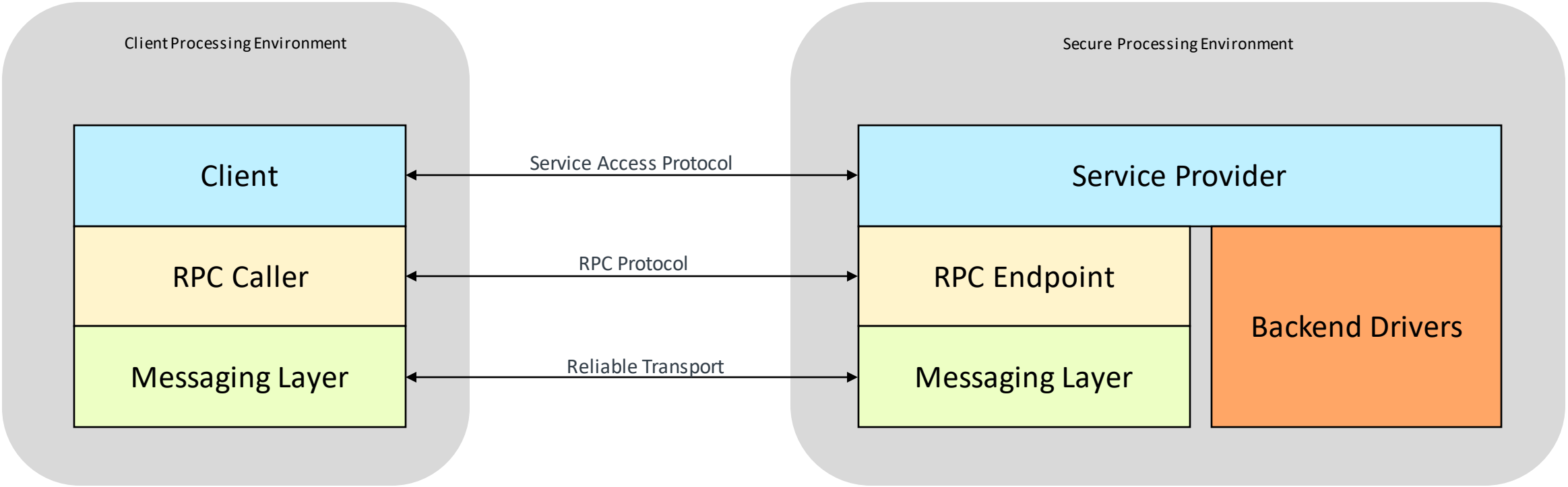
Project Goals

- Adopt a project structure that makes it easy to reuse components.
- Make service interfaces easy to consume by clients.
- Adopt a robust layered model to allow alternative layer implementations to coexist.
- Support service deployment into different processing environments.
- Encourage testing by making it easy to add and run test cases.
- Support test and debug in a native PC environment to help application developers.
- Reuse external projects without having to maintain a fork.
- Provide an extensible build system that can integrate with Yocto or Buildroot based OS builds.

Project Structure



Common Layered Model



arm

Protocols

- Public interface definitions are referred to as *protocols*.
- A service access protocol defines:
  - The set of operations that forms a service interface
  - Per-operation input and output parameters
  - Service specific status codes
- An RPC protocol is responsible for:
  - Qualifying a remote interface instance
  - Qualifying the remote operation to call
  - Forwarding serialized input parameters
  - Returning serialized output parameters
  - Returning generic RPC status
- Protocol definitions are planned to be kept separate from code under the *protocols* repo. This helps to simplify external client project dependencies on the TS project.
- The project structure allows for alternative protocol definition and serialization methods. Currently support:
  - *Protocol Buffers* – language independent interface definition. Convenient for non-C clients.
  - *Packed-C* – extends existing conventions used by SCMI to support variable length parameters.

Client Identity

- A robust identifier for a calling client is important for implementing access control at a service interface.
- For a system partitioned into separate security domains, it should not be possible for a malicious client to fake another client's identity.
- RPC call requests cross execution level boundaries when a call is made from the client security domain to the service provider security domain.
  - Client identity can comprise multiple parts based on perspectives from different execution levels.
- For example, a call request initiated by a client running in a Linux userspace process and destined for a service provider running within an SEL0 SP will traverse at least the following execution levels:
  - Calling process (EL0) -> Kernel (EL1) -> Secure monitor (EL3) -> SPMC (SEL1) -> Service provider (SEL0)
- Client identity information to be added to a call request by higher privilege components in the call path. E.g:
  - Kernel driver adds UID or SELinux label for the calling process
  - Hypervisor/SPMC adds the source partition ID
- A service provider may implement access control policy, using the accumulated client identity as the subject.

Currently supported deployments

Descriptive Name	Environments	Provides	Usage
crypto	opteesp	Crypto primitives with private keystore	General platform service
secure-storage	opteesp	Secure object store	General platform service
component-test	linux-pc, arm-linux	Standalone tests for components and integrations.	Test driven development and regression testing.
ts-service-test	linux-pc, arm-linux	Service interface level end-to-end tests.	Test services from a client perspective.
ts-demo	linux-pc, arm-linux	Demonstration client application.	Example user-space client application.
libts	linux-pc, arm-linux	Provides a uniform interface for locating and accessing services. Decouples an application from any service deployment details.	Client application development and service level testing.
libsp	opteesp	FFA interface library.	Used in SP environments.

Build conventions

- All builds related to deployments use CMake.
- The unit of reuse for source code is referred to as a *component.*
- A *component.cmake* file defines a set of files that can be reused as a unit.
- A CMakeLists.txt file pulls together a set of components and an environment to define an executable or library than can be built and deployed.
- All CMakeLists.txt files live under the *deployments* top-level directory.
- A concrete deployment name is defined by:
  - <descriptive-name>/<environment>
- Example deployment directory structure:
  - ```
    deployments
       |-- ts-service-test
                |-- ts-service-test.cmake
                |-- arm-linux
                |        |-- CMakeLists.txt
                |-- linux-pc
                         |-- CMakeLists.txt
    ```
    ```

Test Conventions

- The TS project has adopted CppUtest for running C/C++ test cases.
- Test components are treated exactly the same as any other source components.
- To reflect the intended subject for tests, test components are located at an appropriate location in the *components* source tree within a subdirectory called *test*.
- For example, service level tests for the crypto service live under:
  - `components/service/crypto/test`
- There are currently two test related deployments:
  - *component-test* – component level tests. Normally built and run as a native PC executable
  - *ts-service-test* – service interface level tests. Either run as a native PC executable or cross compiled to test real service deployments.
- Adding new tests and running them is extremely easy.
  - Adopting a test driven development approach for the majority of components is painless and is warmly encouraged.

Supporting development in a native PC environment

- By their nature, debugging code that runs in a secure processing environment can be tricky.
- As a rule, if a component can be built, run and tested in a native PC environment, it should be.
- Strict conformance to the layered model and the component based organization helps a lot with this.
- Two deployments for test are maintained:
  - *component-test*
  - *ts-service-test*
- Both may be built for the *linux-pc* environment and run from a Linux command prompt.
- Test and debug in the target environment is obviously important but it's not the only option.

Find out more

- Project repo: https://review.trustedfirmware.org/admin/repos/TS/trusted-services
- Docs on readthedocs.org – coming soon (docs can already be built from project repo)
- Contact:
  - julian.hall@arm.com
  - miklos.balint@arm.com
  - gyorgy.szing@arm.com

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה