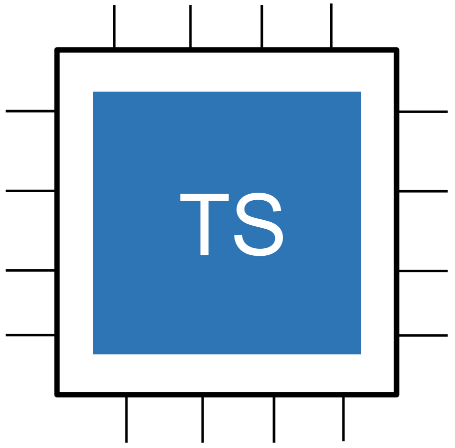
Introduction to the Trusted Services Project

Julian Hall - Arm

Linaro Connect
VIRTUAL • SPRING 2021

A new *trustedfirmware.org* project

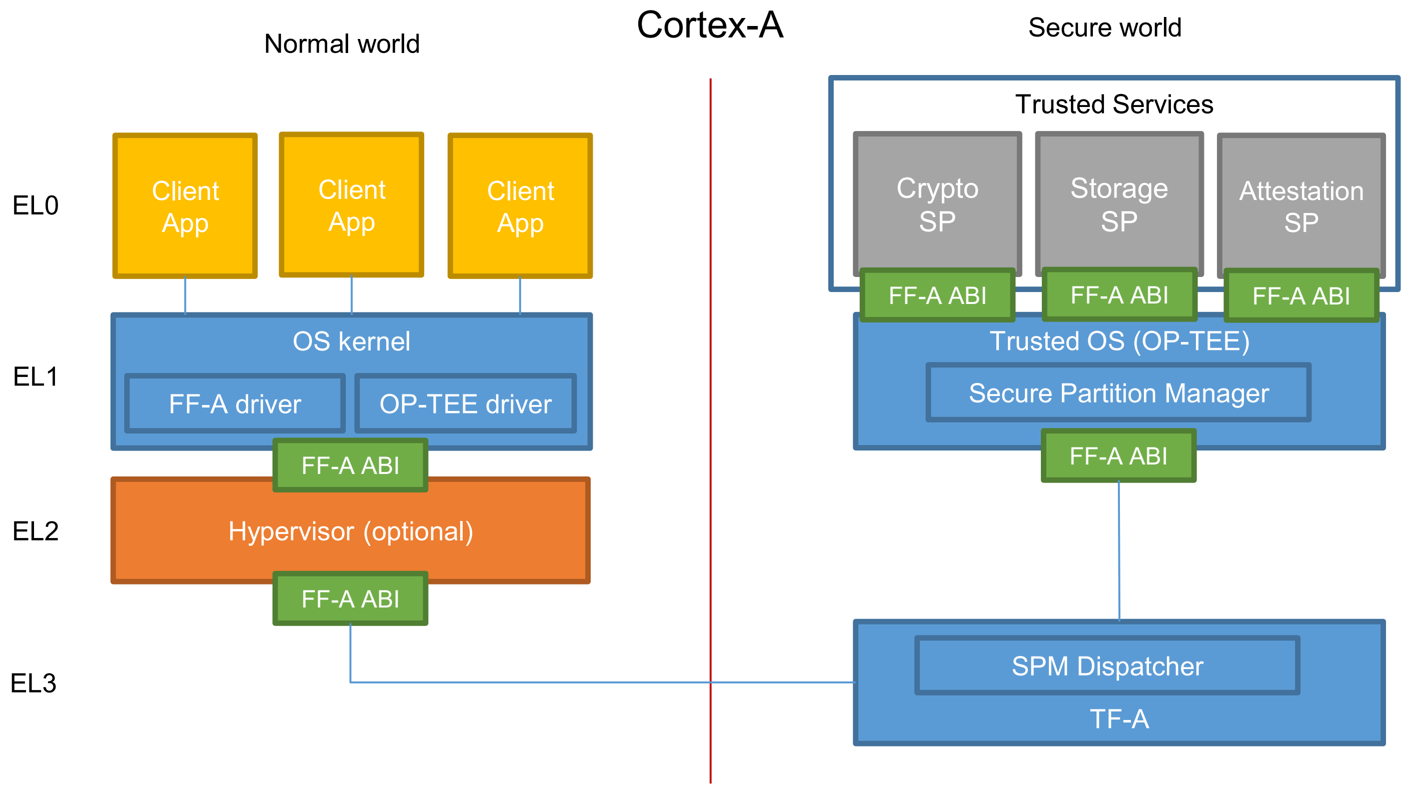


- Trusted Services (TS) is a new project under *trustedfirmware.org*.
- *trustedfirmware.org* background:
  - Open governance community project.
  - Provides reference implementation of secure world firmware for Armv8-A and Armv8-M.
  - Provides SoC developers and OEMs with a reference code base that complies with Arm specifications.
- First published to TrustedFirmware git on 26th November 2020.
- Originates from work by the Arm OSS firmware group to provide PSA services on Cortex-A.
- Services help meet PSA certification requirements.
- Complements existing *trustedfirmware.org* projects.

What are Trusted Services?

- A general term for a firmware service that performs security related operations on behalf of clients.
- Clients could be:
  - User-space applications
  - Kernel drivers
  - Other trusted services
- 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 a Secure Partition Manager (SPM)
  - Trusted applications – managed by a TEE
  - Secure enclave – a secondary MCU
- Example services:
  - Crypto – provides cryptographic operations with a protected key store
  - Protected storage – provides protected persistent storage
  - fTPM – TPM 2.0 firmware, running as a trusted service
  - UEFI Keystore – a protected persistent store for UEFI keys

Reference Deployment



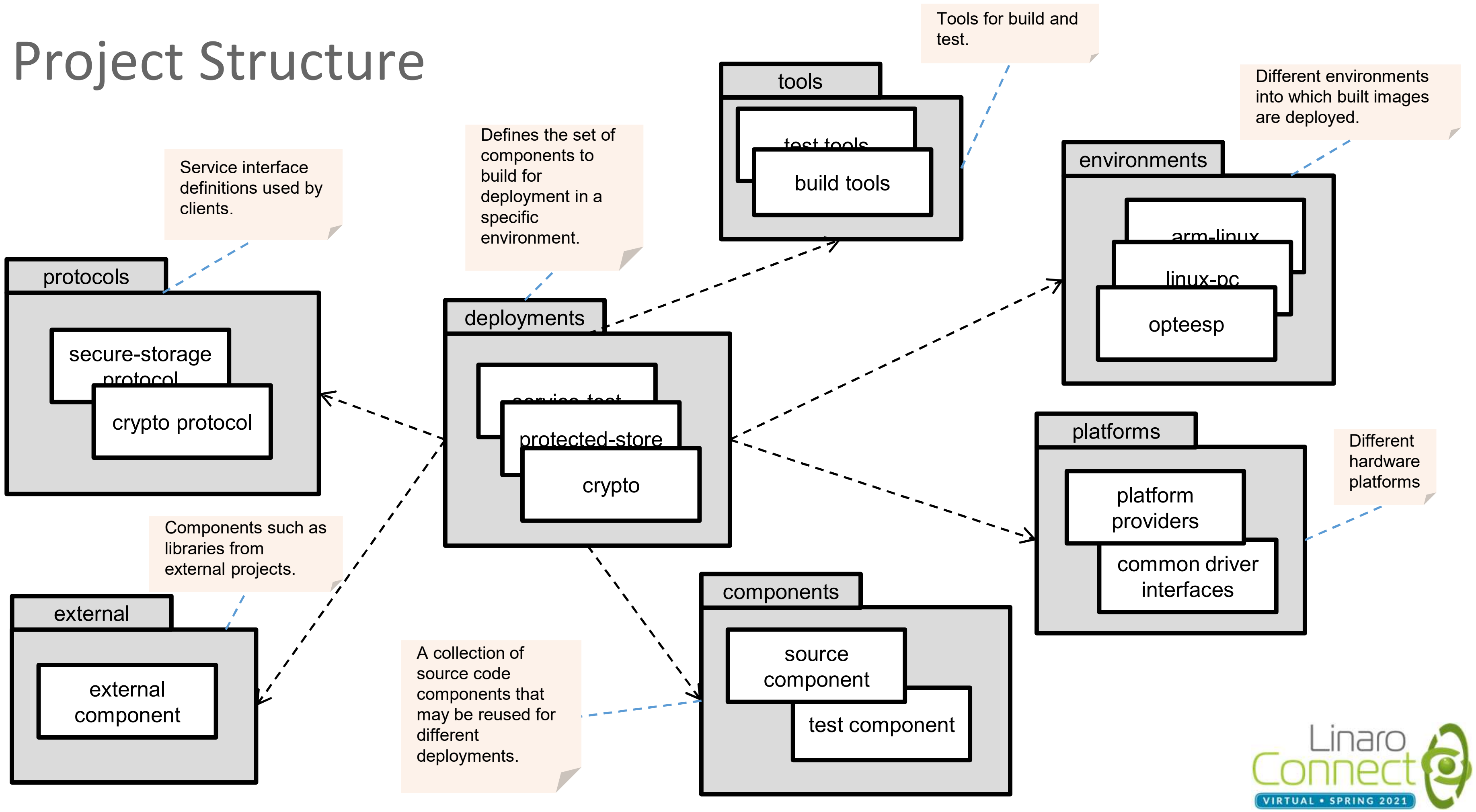
Why have a separate Trusted Services project?

- 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.
- 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.
- Creates opportunities for running trusted services on any 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 components from external projects without having to maintain a fork.
- Enable SoC and platform developers to contribute hardware specific code.
- Provide an extensible build system that can integrate with OS build systems such as Yocto or Buildroot.

Project Structure



Tools for build and test.
tools
test tools
build tools
Different environments into which built images are deployed.
environments
arm-linux
linux-pc
opteesp
Service interface definitions used by clients.
Defines the set of components to build for deployment in a specific environment.
protocols
secure-storage protocol
crypto protocol
deployments
service-test
protected-store
crypto
platforms
platform providers
common driver interfaces
Different hardware platforms
Components such as libraries from external projects.
external
external component
A collection of source code components that may be reused for different deployments.
components
source component
test component

Building Trusted Services - *deployments*

- A build of a binary, elf file, library (or whatever) is referred to as a *deployment*.
- All deployment build files live under the *deployments* top-level directory.
- A concrete deployment name is defined by:
    - <descriptive-name>/<environment> + <platform> (if necessary)
- All deployment builds 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 for a deployment combines a set of components and an environment to define an executable or library than can be built and used.

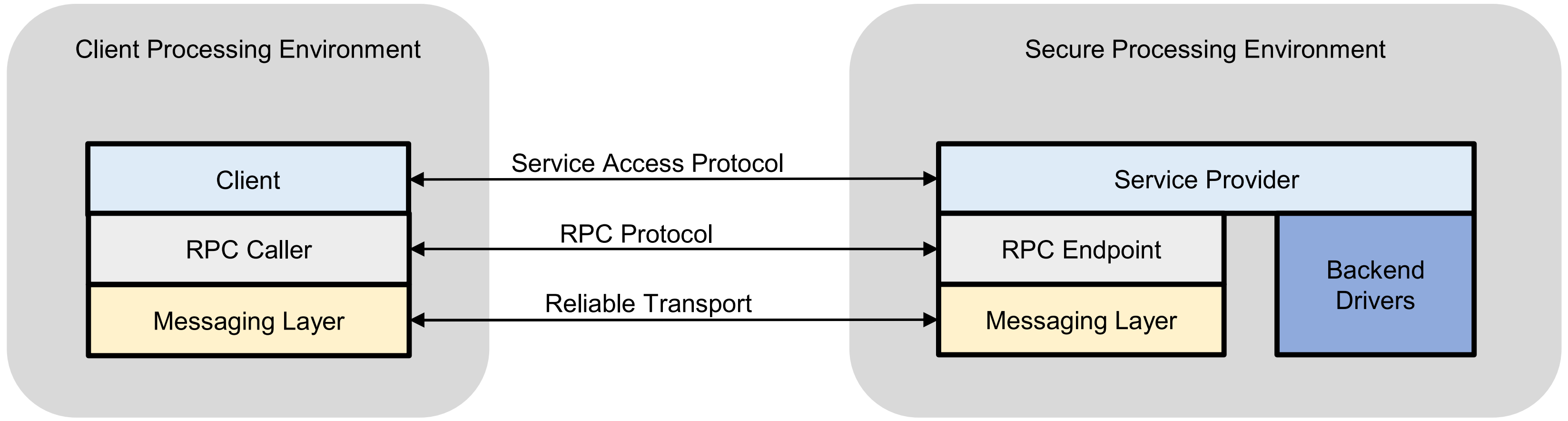
- Example deployment directory structure:

    -
      ```
      deployments
      |-- ts-service-test
            |-- ts-service-test.cmake
            |-- arm-linux
            |       |-- CMakeLists.txt
            |-- linux-pc
                    |-- CMakeLists.txt
      ```

Using Trusted Services - *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:
  - Making the remote call
  - Returning the result
- 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.

Common Layered Model



- Layered model reflected in project structure.
- Keeping layer dependencies as simple as possible helps with reuse and extending for new services and environments.

Test Strategy

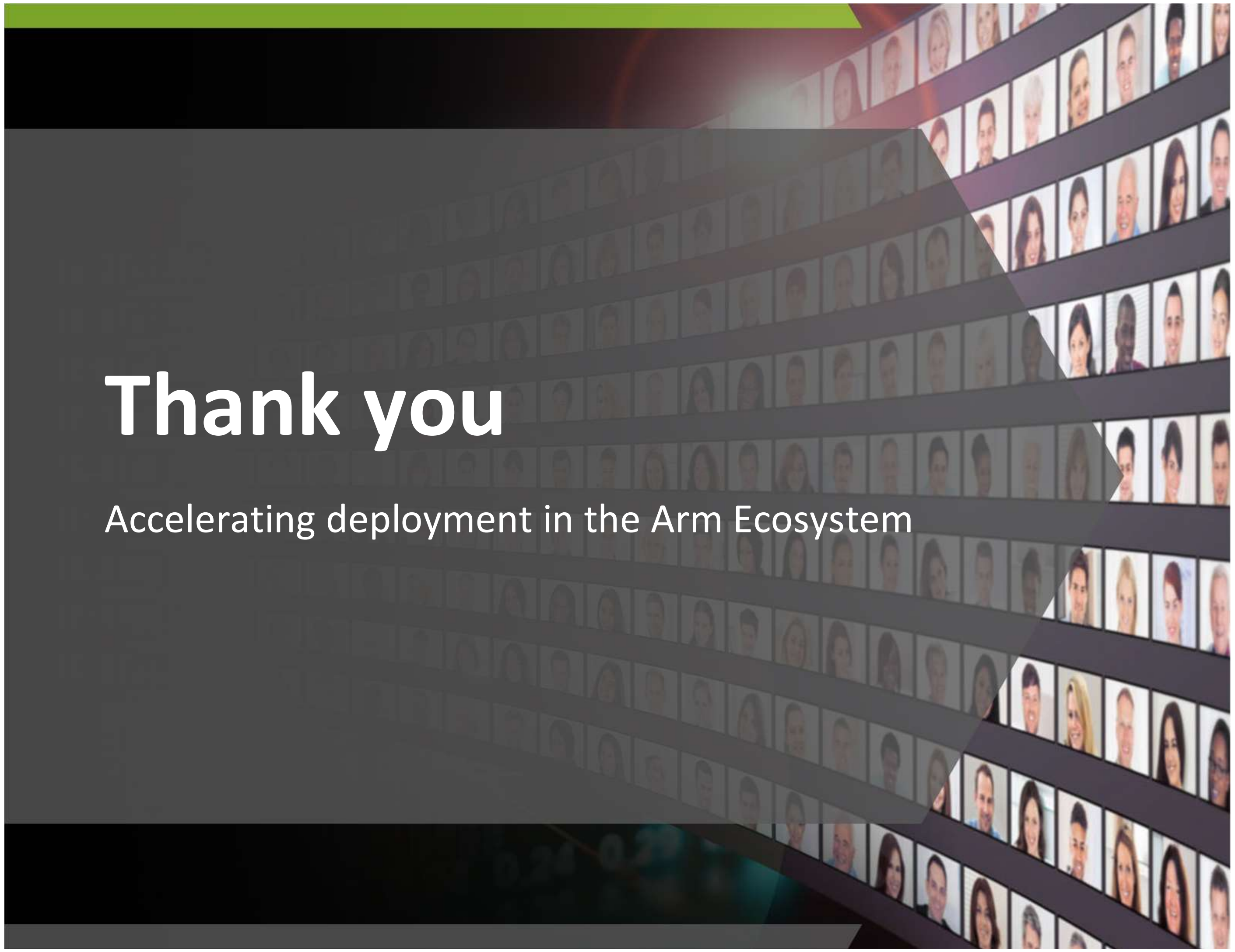
- Testing incorporated into project from day one.
- Different classes of test:
  - *Unit tests* – sub-component level tests.
  - *Component tests* – tests individual and assemblies of components.
  - *Service tests* – end-to-end service tests using standard access protocols.
  - *Environment tests* – tests low-level services provided by a secure processing environment.
- The TS project has adopted CppUtest for running C/C++ test cases.
- Many tests may be run in a native PC environment:
  - Quick and convenient way of checking for regressions.
  - Debug more straight forward in PC environment.
- Adding new tests and running them is extremely easy.

Future steps

- Add support for real hardware platforms – currently support Arm FVP only.
  - Allow platform support model to be pipe cleaned
- Add new services – currently support Crypto and Secure Storage.
- Harden security – in particular, client identity and access control.
- PSA level 1 and level 2 certify hardware platforms.
  - Will enable SoC developers and OEMs to use TS as PSA RoT on their platforms
- Extend testing to better represent real-life use-cases.

Find out more

- Visit: https://www.trustedfirmware.org/projects/trusted-services
- Project repo: https://git.trustedfirmware.org/TS/trusted-services.git
- Docs: https://trusted-services.readthedocs.io/en/latest/index.html
- Mailing list: https://lists.trustedfirmware.org/mailman/listinfo/trusted-services



Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect VIRTUAL • SPRING 2021