Parsec – Platform Abstraction for Security

Project Introduction For Yocto Virtual Summit
May 26th 2021
Speakers

Paul Howard
Principal System Solutions Architect
paul.howard@arm.com
@paulhowardarm

Anton Antonov
Senior Engineer
anton.antonov@arm.com
@anta5010
Agenda

- What is Parsec?
- Using Parsec
- Details of Parsec/Yocto Integration
- Questions
Parsec: A Platform Abstraction For Security

- TRUSTED PLATFORM MODULE (TPM)
- HARDWARE SECURITY MODULE (HSM)
- SECURE ELEMENTS
- SERVICES IN TRUSTED ENVIRONMENTS

- PORTABLE
- CONVENIENT
- MULTI-TENANT
- CLOUD-NATIVE
Service Architecture

Cloud-native delivery/orchestration

Wire protocol based on PSA Crypto API

Platform-Agnostic

IPC

Parsec Service

TPM

HSM

TEE

Trusted App

Application

Client Library

Listener

Provider
Parsec in the Security API Landscape

Additional Convenience Features and Identity Management

Multi-Language Layer

Core API Contracts Based On Strongly-Specified, Modern PSA Standards

Rust/C Interop Through Limited Number of Well-Audited Call Points

C Language Layer

TPM 2.0
- Hardware TPM
- Firmware TPM

Oasis PKCS#11
- HSM
- SmartCard
- Shims to Other Technologies...

Vendor-Specific (eg. CryptoAuthLib)
- Secure Element
- Custom HW/FW

PSA Functional API
- PSA Root of Trust

Core API Contracts Based On Strongly-Specified, Modern PSA Standards
The Growing Ecosystem
Why Add Parsec To Yocto?

- Architecture Neutral
- Supports Diverse Hardware Through Customization
- Common Developer Experience Across Platforms
  - Commitment to Open Development
  - Targeting IoT/Embedded/Edge Space
- Complements Packaging For Off-The-Shelf Distros

Meta-parsec is a sublayer of meta-security since Hardknott

https://git.yoctoproject.org/cgit/cgit.cgi/meta-security/tree/meta-parsec
How To Use Parsec In Yocto

**Build**
Include `meta-parsec, meta-rust and meta-clang` in your layers list. Include `parsec-service` into your image.

**Configure**
Parsec is configured simply with a TOML file. Examples are provided to connect the service with TPM, HSM/PKCS#11 or software back-ends.

**Run**
The Parsec service is a single executable that runs locally. It can be managed with `systemd` (recommended), or SysV init scripts.

**Consume**
Use the command-line `parsec-tool` if desired or consume the APIs into your code from languages including Rust, C and Go, with more to come...

https://git.yoctoproject.org/cgit/cgit.cgi/meta-security/tree/meta-parsec/README.md
The Developer Experience: Command-Line Example

```bash
[hugdev01@machine ~]$ ./parsec-tool create-ecc-key -k "rusty key 🌐"
[INFO ] Creating ECC key...
[INFO ] Key "rusty key 🌐" created.
[hugdev01@machine ~]$ ./parsec-tool sign -k "rusty key 🌐" "Cloud Native Rust Day"
[INFO ] Hashing data with Sha256...
[INFO ] Signing data with Ecdsa { hash_alg: Specific(Sha256) }...
MEUCIQdG4leLYVBTEd11J3I5Lukaf7XBa5+HLK+9aVG4730VAIgWP6JRGKyp500oCofQ+20v8SvM9VaJRfBMcvAW/DnVy0=
[hugdev01@machine ~]$ ./parsec-tool export-public-key -k "rusty key 🌐"
-----BEGIN PUBLIC KEY-----
BPtwNlxMRHSrkSZGkBLU7mPcT2Dc4bVePOFvxX/FFH1cYN6IUBlvqCqpK0v2VuDN
TIipHdxoXjoXQxpD2Nczo0=
-----END PUBLIC KEY-----
```
fn main() {
    use parsec_client::core::interface::operations::{psa_algorithm, psa_key_attributes::{Attributes, EccFamily, Lifetime, Policy, Type, UsageFlags,}};

    let key_name = String::from("rusty key ");
    let alg = psa_algorithm::AsymmetricSignature::Ecdsa {
        hash_alg: psa_algorithm::Hash::Sha256.into(),
    };
    let key_attrs = Attributes {
        lifetime: Lifetime::Persistent,
        key_type: Type::EccKeyPair {
            curve_family: EccFamily::SecpR1,
        },
        bits: 256,
        policy: Policy {
            usage_flags: UsageFlags {
                sign_hash: true,
                ..Default::default()
            },
            permitted_algorithms: alg.into(),
        },
    };

    let client = parsec_client::BasicClient::new(None).unwrap();
    client.psa_generate_key(key_name.clone(), key_attrs).unwrap();
    client.psa_sign_hash(key_name.clone(), b"Cloud Native Rust Day", alg).unwrap();
    let _public_key = client.psa_export_public_key(key_name).unwrap();
}
Some Details of Parsec Integration into Yocto

See also “Using Rust with bitbake and meta-rust” with Steven Walter, 14:45 UTC (Presentation Room)
## Choices For Including Rust-Based Software in Yocto

### Toolchain

<table>
<thead>
<tr>
<th>meta-rust</th>
<th><a href="https://github.com/meta-rust/meta-rust">https://github.com/meta-rust/meta-rust</a></th>
</tr>
</thead>
</table>
| • Builds rust compiler and cargo build system from source  
  • Provides “crate” fetch mechanism for dependencies |

<table>
<thead>
<tr>
<th>meta-rust-bin</th>
<th><a href="https://github.com/rust-embedded/meta-rust-bin">https://github.com/rust-embedded/meta-rust-bin</a></th>
</tr>
</thead>
</table>
| • Uses pre-built upstream versions of compiler and cargo  
  • Faster to build, but less flexible |

### Dependency Management

<table>
<thead>
<tr>
<th>bitbake vending</th>
<th><a href="https://github.com/meta-rust/cargo-bitbake">https://github.com/meta-rust/cargo-bitbake</a></th>
</tr>
</thead>
</table>
| • Dependencies modelled explicitly in the recipe  
  • Needs to be kept in sync with Cargo.toml  
  • Tools to auto-generate include files |

<table>
<thead>
<tr>
<th>cargo vending</th>
<th></th>
</tr>
</thead>
</table>
| • Cargo build system fetches crates by itself  
  • Add CARGO_DISABLE_BITBAKE_VENDORING = "1" to recipe |
Parsec Service Recipe (Fragment)

```bash
inherit cargo
SRC_URI += "crate://crates.io/parsec-service/${PV} \
"

CARGO_BUILD_FLAGS += " --features all-providers,cryptoki/generate-bindings,tss-esapi/generate-bindings"
DEPENDS = "tpm2-tss"
TOOLCHAIN = "clang"
PARSEC_CONFIG ?= "${S}/config.toml"

do_install_append () {
  install -d -m 700 -o parsec -g parsec "${D}${libexecdir}/parsec"
  install -m 700 -o parsec -g parsec "${WORKDIR}/build/target/${CARGO_TARGET_SUBDIR}/parsec" ${D}${libexecdir}/parsec/parsec
}
require parsec-service_${PV}.inc

parsec-service_.bbappend

FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"SRC_URI += "file://config-tpm.toml \
"
PARSEC_CONFIG = "${WORKDIR}/config-tpm.toml"
```
Rust Recipes In CI Pipelines

- Rust tool chain requirements can result in lengthy image build
- Use of persistent **SSTATE_DIR** and **DL_DIR** recommended
- See https://git.yoctoproject.org/cgit/cgit.cgi/meta-arm/tree/.gitlab-ci.yml

```yaml
setup:
  stage: build
  variables:
    KAS_REPO_REF_DIR: $CI_BUILDS_DIR/persist/repos
    SSTATE_DIR: $CI_BUILDS_DIR/persist/sstate
    DL_DIR: $CI_BUILDS_DIR/persist/downloads

before_script:
  - echo SSTATE_DIR = $SSTATE_DIR
  - echo DL_DIR = $DL_DIR
```
Learn More

Get the code

https://github.com/parallaxsecond/parsec

Read the book

https://parallaxsecond.github.io/parsec-book
Join The Community

https://github.com/parallaxsecond/community

#parsec on CNCF https://slack.cncf.io

Every Tuesday at 16:30 (UK), 11:30 (US East), 08:30 (US West)