# FIDO Device Onboarding

# Late-binding Provisioning & Tales from the Trenches of Bleeding Edge Tech

Yocto Summit 2023

Tymoteusz Burak







# Tymoteusz Burak *Junior Embedded Systems Developer*

- <u>symoteusz.burak@3mdeb.com</u>
- linkedin.com/in/tymoteuszburak-a108252a0

- 6 months in 3mdeb
- Integration of functionalities and the creation of Operating Systems for embedded devices in Yocto
- Working on my Bachelor's Degree in Automation and Robotics













- coreboot licensed service providers since 2016 and leadership participants
- UEFI Adopters since 2018
- Yocto Participants and Embedded Linux experts since 2019
- Official consultants for Linux Foundation fwupd/LVFS project since 2020
- IBM OpenPOWER Foundation members since 2020



- What is FDO?
- Existing implementations of FDO protocol
- Challenges faced in integrating the <u>fido-device-onboard-rs</u> project in Yocto
  - Current Rust implementations inside of Yocto
  - Bitbake environment vs pkg-config
- Demo presentation
- Q&A







An automatic onboarding protocol for IoT devices. Permits late binding of device credentials, so that one manufactured device may onboard, without modification, to many different IOT platforms\*

FIDO® is a trademark (registered in numerous countries) of FIDO Alliance, Inc All trademarks used are properties of their respective owners

<sup>\*</sup> quote from FIDO Device Onboard Specification 1.1





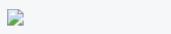


#### Example stakeholders:

- Distributors
- Retailers
- System Integrators
- Certification Agencies



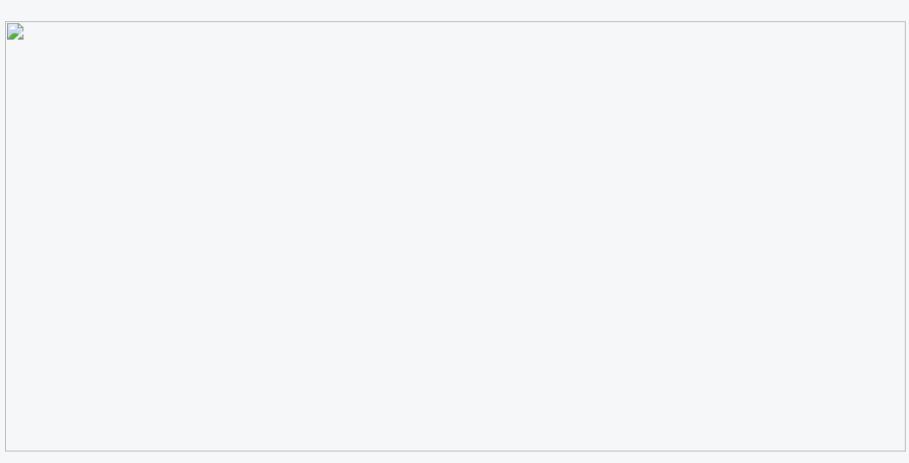




















## Existing implementations of FDO protocol

### FDO Project

- <u>client-sdk-fidoiot</u>
- pri-fidoiot



## Existing implementations of FDO protocol



### fido-device-onboard-rs

- fdo-client-linuxapp
- fdo-rendezvous-server
- fdo-owner-onboarding-server
- fdo-serviceinfo-api-server

- fdo-manufacturing-client
- fdo-manufacturing-server
- fdo-owner-tool





"That should be easy right?"





#### meta-rust

- Source-based, thus more customizable
- Allows for an offline build via cargo-bitbake
- Doesn't contain cross or rustcnightly

#### meta-rust-bin

- Provides pre-built compiler and cargo
- Needs online build
- Allows for the use of rustcnightly and cross



## pkg-config

"The pkg-config command usually doesn't support cross-compilation, and this crate prevents it from selecting incompatible versions of libraries. Setting PKG\_CONFIG\_ALLOW\_CROSS=1 disables this protection, which is likely to cause linking errors, unless pkg-config has been configured to use appropriate sysroot and search paths for the target platform."

docs.rs/pkg\_config



```
# By default pkg-config variables point to aarch64 libraries which are picked up
# during x86 64 builds, this causes aarch64 include directories and linker
# search paths to into x86 64 builds, causing problems.
# Host libraries already use absolute paths so set sysroot to /
export PKG CONFIG SYSROOT DIR="/"
export PKG_CONFIG_PATH="${RECIPE_SYSROOT_NATIVE}/usr/lib/pkgconfig:${RECIPE_SYSROOT_NATIVE}/usr/share/pkgconfig"
export PKG CONFIG LIBDIR="${RECIPE SYSROOT NATIVE}/usr/lib/pkgconfig"
export PKG CONFIG DIR="${RECIPE SYSROOT NATIVE}/usr/lib/pkgconfig"
# Those variables are handled internally by pkg-config crate.
# All paths are relative to sysroot, so set PKG CONFIG SYSROOT DIR
# The PKG CONFIG * {TARGET} needs underscores in it's triple instead of hyphens
export PKG CONFIG TARGET VAR = "${@d.getVar('TARGET SYS').replace('-',' ')}"
do compile:prepend() {
    export PKG_CONFIG_SYSROOT_DIR_${PKG_CONFIG_TARGET_VAR}="${RECIPE SYSROOT}"
    export PKG CONFIG PATH ${PKG CONFIG TARGET VAR}="${RECIPE SYSROOT}/usr/lib/pkgconfig:${RECIPE SYSROOT}/usr/share/pkgconfig:$
    export PKG_CONFIG_LIBDIR_${PKG_CONFIG_TARGET_VAR}="${RECIPE_SYSROOT}/usr/lib/pkgconfig"
    export PKG CONFIG DIR ${PKG CONFIG TARGET VAR}="${RECIPE SYSROOT}/usr/lib/pkgconfig"
```



### openssl-kdf

```
let kdf_h_cts = std::fs::read_to_string("/usr/include/openssl/kdf.h").unwrap();
```



```
fn read_header(lib: &pkg_config::Library, path_rel: &str) -> std::io::Result<String> {
    for dir in lib
        .include_paths
        .iter()
        .map(|p| p.as_path())
        .chain(std::iter::once(std::path::Path::new("/usr/include")))
    {
        match std::fs::read_to_string(dir.join(path_rel)) {
            Ok(r) => return Ok(r),
            Err(e) if e.kind() == std::io::ErrorKind::NotFound => continue,
            Err(e) => return Err(e),
        }
    }
}

return Err(std::io::ErrorKind::NotFound.into());
}
```

```
- let openssl_version = openssl.version;
+ let openssl_version = &openssl.version;
```

```
- let kdf_h_cts = std::fs::read_to_string("/usr/include/openssl/kdf.h").unwrap();
+ let kdf_h_cts = read_header(&openssl, "openssl/kdf.h").unwrap();
```



#### devicemapper-sys



## Live Demo





3mdeb/meta-fdo

Image used for demo



- https://fidoalliance.org/
- <a href="https://fido-device-onboard.github.io/docs-fidoiot/latest/">https://fido-device-onboard.github.io/docs-fidoiot/latest/</a>
- <a href="https://www.lfedge.org/projects/fidodeviceonboard/">https://www.lfedge.org/projects/fidodeviceonboard/</a>
- https://docs.rs/pkg-config/latest/pkg\_config/

#### Contact

- <u>leads@3mdeb.com</u>
- <u>w tymoteusz.burak@3mdeb.com</u>



