NuttX 2019 International workshop

FEIG ELECTRONIC

NUTTX AND PAYMENT CARD INDUSTRY SECURITY STANDARDS

Michael Jung

michael.jung@feig.de / mijung@gmx.net

FEIG ELECTRONIC GmbH
D-35781 Weilburg, info@feig.de, www.feig.de

SESSION OBJECTIVES

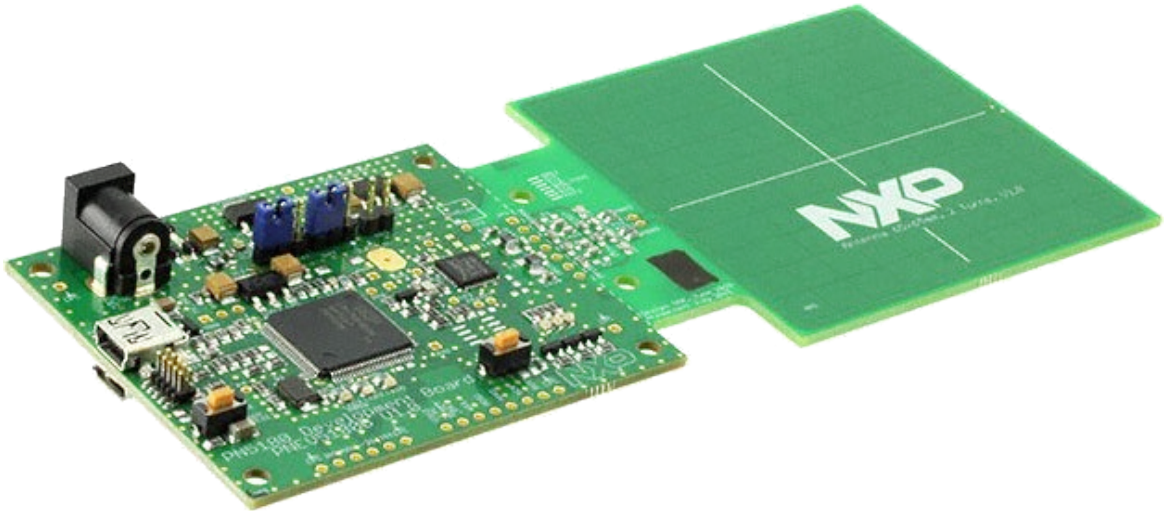
- Discuss Approaches and Get Community Feedback for
- Enhancing NuttX Security w.r.t.
  - Cryptographic Key Storage and Access Control
  - Code Signing and NuttX SDKs
- To Empower NuttX as a Platform for
  - Payment Devices
  - Secure IoT Devices

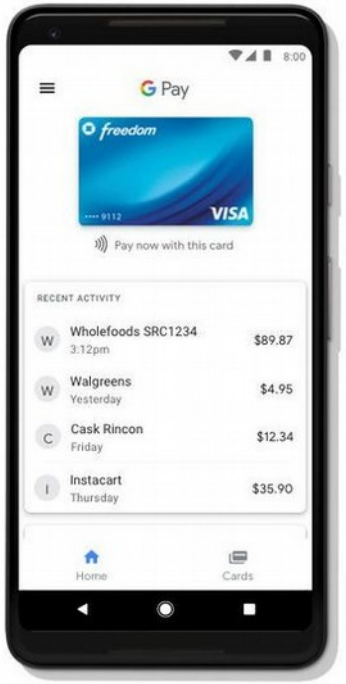
FEIG ELECTRONIC - CVEND

PAYMENT

- Linux® based Contactless (NFC) Secure Card Reader (SCR)
- EMV® Contactless Kernels for all major Credit Card Brands
- Tamper Responsive Cryptographic Tokens with PKCS#11 API
- Secure Boot, Secure Firmware Update, Signed Application Code
- Payment Applications developed by Third Partys with FEIG cVEND SDK: GNU Toolchain, FEIG Libraries and Tools
- Payment Card Industry - PIN Transaction Security - Point of Interaction (PCI PTS POI) Version 4.0 Compliant
- Card Data Protection only (No PIN Processing)
- “The Raspberry Pi® of Payment Terminals” - Good Market Reception

COULD THIS BE DONE WITH NUTTX ON AN MCU? DEMO!



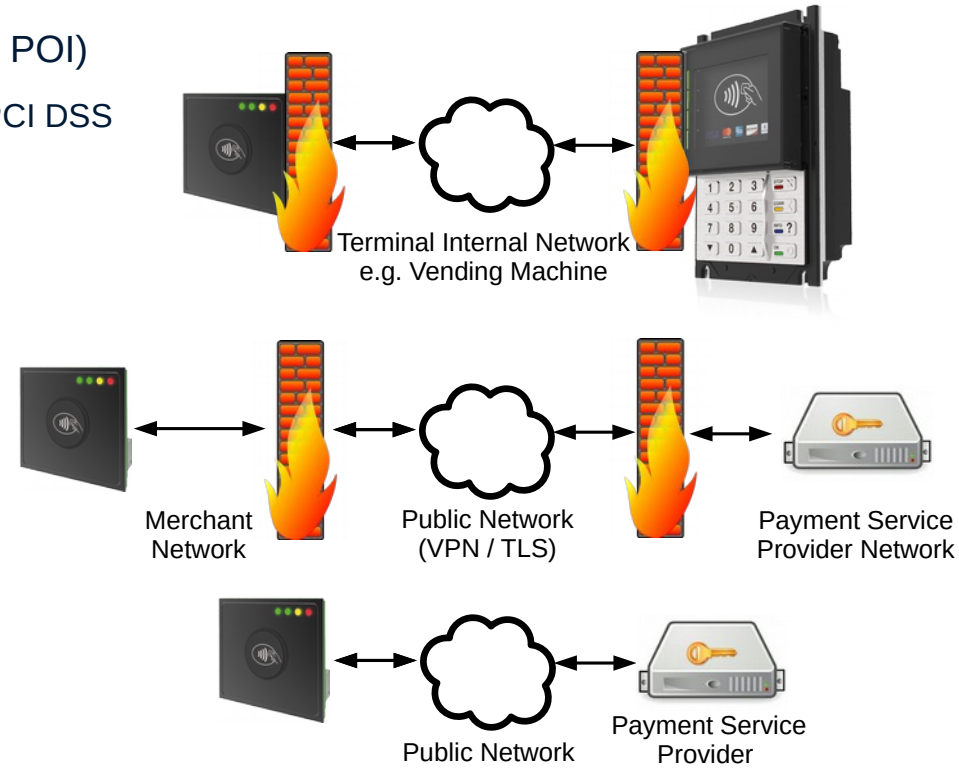


MINIMAL POINT-OF-SALE DEMO

PAYMENT CARD INDUSTRY SECURITY STANDARDS

PCI Security Standards Council

- PIN Transaction Security - Point of Interaction (PCI PTS POI)
  - Not strictly required if no PINs are handled, but helps for PCI DSS
  - SCR Requirements: Keystore / No Application
- PCI Data Security Standard (PCI DSS)
  - Assessing the complete Card Data Environment (CDE)
  - No Secure Card Reader required
- PCI Point-to-Point-Encryption (PCI P2PE)
  - Terminal-to-Host-Encryption
  - Merchant Network no longer in scope of PCI DSS
  - Requires Host specific Protocol Encryption on Terminal
  - SCR Requirements: Keystore and Application



CRYPTOGRAPHIC KEY STORAGE AND ACCESS CONTROL

How to guarantee that applications

- never have access to clear text cryptographic keys, and
- can use cryptographic keys only for their intended purpose

PROTECTING KEYS WITH AN APPLICATION INTERPRETER

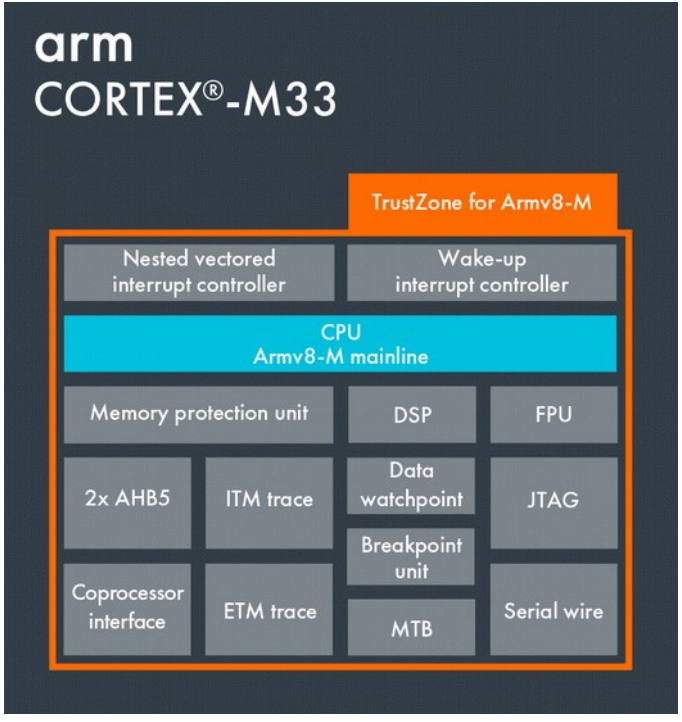
- Use MicroPython, QuickJS, or other interpreters
- Controlled Access to Keystore via C function wrappers
  - MicroPython user C module, or
  - QuickJS C API, or
  - ...
- NuttX Flat Build sufficient (e.g. no Memory Protection required)
- Cons:
  - Considerable Resource Load
  - Embedded Developers love C

PROTECTING KEYS WITH THE MEMORY PROTECTION UNIT

- NuttX Protected Build
  - Keystore “Device Driver” to provide Crypto Services and enforce Access Control
  - Keys stored in MPU protected privileged RAM / Flash / Battery-Backed SRAM
  - PKCS#11 API (or conceptually similar) via `ioctl()`
  - Might be Sebastien Lorquet’s Crypto Mangager
- Just how secure is the NuttX Protected Build? From the NuttX TODO file:
  - “In the current design, the kernel code calls into the user-space allocators to allocate user-space memory.”
    - At least to allocate space for a new tasks stack. Others? “That could be fixed by dropping to user mode”. Hard?
  - “Another place where the system calls into the user code in kernel mode is work_usrstart() to start the user work queue.”
    - Plugged by de-configuring LIB_USRWORK?
  - “When a C++ ELF module is loaded, its C++ constructors are called via sched/task_starthook.c logic. This logic runs in protected mode.”
    - Plugged by BINFMT_DISABLE?
  - Are there more known holes to be plugged?

PROTECTING KEYS WITH ARM TRUSTZONE

arm TRUSTZONE



- Upcoming Cortex-M23 / -M33 MCUs include TrustZone technology
  - E.g. STM32L5 or LPC5500
    - Focus on IoT Security
    - Sampling now
    - ARMv8-M
- NuttX as a Trusted Execution Environment?
  - “SMP and TrustZone on the i.MX6 quad was part of a research project with a University [...]”

    ”I have heard of people using NuttX as TrustZone masters on high end products [...]”

    (Greg’s comment on NuttX Issue #92)
  - There are some references to TRUSTZONE in `arch/arm/src/armv7-a/arm_gicv2.c`

NuttX 2019 International workshop

FEIG ELECTRONIC

NUTTX BASED SOFTWARE DEVELOPMENT KIT

How to ship an embedded device with

- vendor controlled firmware (including the NuttX kernel)
- on which system integrators can install their own application

BOARD SPECIFIC NUTTX SOFTWARE DEVELOPMENT KIT

- Configure **`nuttx`** for Protected Mode Kernel and SDK build for a certain board (e.g. **`pnev5180b/sdk`**)
- Create GNU toolchain with Buildroot (e.g. **`cortexm3-eabi-defconfig-7.3.0`**)
- **`make pass2`** generates the ***kernel blob*** (to be flashed via e.g. DFU as *firmware*)
- **`make pass1 export`** creates **`nuttx-export.zip`**
  - copy header files and **`libnuttx.a`** into toolchain
  - copy linker script (**`nuttx.ld`**) and **`userspace.c`** into toolchain
- Example: Compile hello world example into a ***userspace blob*** (to be flashed via e.g. DFU as *application):*
  - **`arm-nuttx-eabi-gcc -o hello hello.c ~/nuttx-sdk/share/nuttx/userspace.c -nostdlib -lnuttx -Tldscripts/nuttx.ld`**
- Streamlining: Compile **`userspace.c`** into **`libnuttx.a`**, make **`nuttx.ld`** default linker script, make **`-nostdlib`** implicit. Ideas?
- With a **`config.site`** file in the **`sysroot`** static libraries can be compiled for NuttX from autotools packages.
- Buildroot with NuttX kernel? http://lists.busybox.net/pipermail/buildroot/2015-July/131978.html

NuttX 2019 International workshop

FEIG ELECTRONIC

SECURE BOOT AND CODE SIGNING

How to guarantee that access to the system integrator‘s cryptographic keys is granted only to his application

SECURE BOOT / SIGNED APPLICATIONS

- Integrity Check of Kernel Blob Out-of-Scope
  - Boot ROM's or 1st Level Boot Loader's job
- Integrity Check of Application Code (Only Authorized Applications get Keystore Access)
  - Extend the User-Mode Blob Header (`struct userspace_s`) with meta-data:
    - A digital signature (e.g. RSASSA-PKCS1-v1_5) of `us_entrypoint` value and `text`, `data`, `rodata`, and ~~`bss`~~ section contents.
    - A version field to protect against downgrades.
- `nx_start_application()` verify that
  - no downgrade was performed (by comparing version field agains value stored in Flash / EEPROM of highest ever installed application code version), and
  - the digital signature is correct with a Public Key stored as part of the NuttX Kernel Blob or in the Keystore
- If checks fail stay in Device Firmware Upgrade (DFU) mode

- There Probably are Questions
- But they are not on this Bulleted List
- Because of Causality

CREDITS

- Title Page Photo by Florian Klauer on Unsplash
- Objectives Slide Photo by Alexander Muzenhardt on Unsplash
- Key Storage Slide Photo by James Sutton on Unsplash
- Code Signing Slide Photo by Helloquence on Unsplash
- SDK Slide Photo by Markus Spiske on Unsplash
- Thanks Slide Photo by Courtney Hedger on Unsplash
- Causality joke on Thanks Slide by Joseph Poon and Thaddeus Dryja
- All trademarks cited are the property of their respective owners