RISC-V Workshop
Secure Bootloader

June 11, 2019
Technical Workshop

David Garske, Software Engineer
Daniele Lacamera, Software Engineer

Outline

A. Introduction
B. Secure Boot for Microcontrollers
C. wolfBoot Overview & Design
D. wolfBoot RISC-V port

Introduction

wolfSSL Company

wolfSSL Introduction

- Open source software company
- Founded in 2004 by our CEO Larry Stefonic and CTO Todd Ouska
- Commercial grade security libraries
- Designed to support embedded use
- Dual license GPLv2 or Commercial
- Products (partial list):
  - wolfSSL: TLS (SSLv3.0 - TLSv1.3)
  - wolfCrypt: Cryptographic algorithms
  - wolfBoot: Secure Bootloader
  - wolfSSH: SSH Server and Client
  - wolfMQTT: MQTT v3.1.1 - v5 and MQTT-SN
  - wolfTPM: TPM 2.0 Hardware

Secure Boot for Microcontrollers

Requirements & Goals

Secure Bootloader Requirements

How to implement bootloader / firmware update?

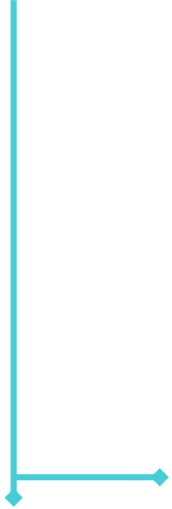
The IETF SUIT group is currently working on an internet draft [1] “A Firmware Update Architecture for Internet of Things Devices”:

- “The bootloader must be **minimal**, containing only flash support, cryptographic primitives and optional recovery mechanism.” (3.6)
- Remote **firmware transfer** is implemented in the application, as it is specific to the connectivity technology in use.
- The bootloader must provide a secure mechanism based on **asymmetric cryptography**, to verify the authenticity and the integrity of the firmware.

[1] https://datatracker.ietf.org/doc/draft-ietf-suit-architecture/

Secure Bootloader Goals

- Portability across all microcontrollers and operating systems
- Public key based authentication to verify firmware
- Small footprint
- Minimal external dependencies
- Memory safety (no malloc/free)
- Reliable Firmware update mechanism
  - Independent from the application
  - Supports fallback to previous version if update fails



wolfBoot

Overview & Design

wolfBoot: Secure Bootloader

- The Secure Bootloader for microcontrollers
  - Runs on virtually any 32 bit microcontroller
  - Multi-architecture support:
    - RISC-V E31
    - ARM Cortex-M[0-7] / Cortex-M33/M23
    - MIPS32 (coming soon)
- OS-Agnostic: runs with any existing bare-metal or RTOS solution
- Firmware verified at boot time using wolfCrypt ECDSA or ED25519
- No additional hardware required
- Source code and firmware update examples available on GitHub

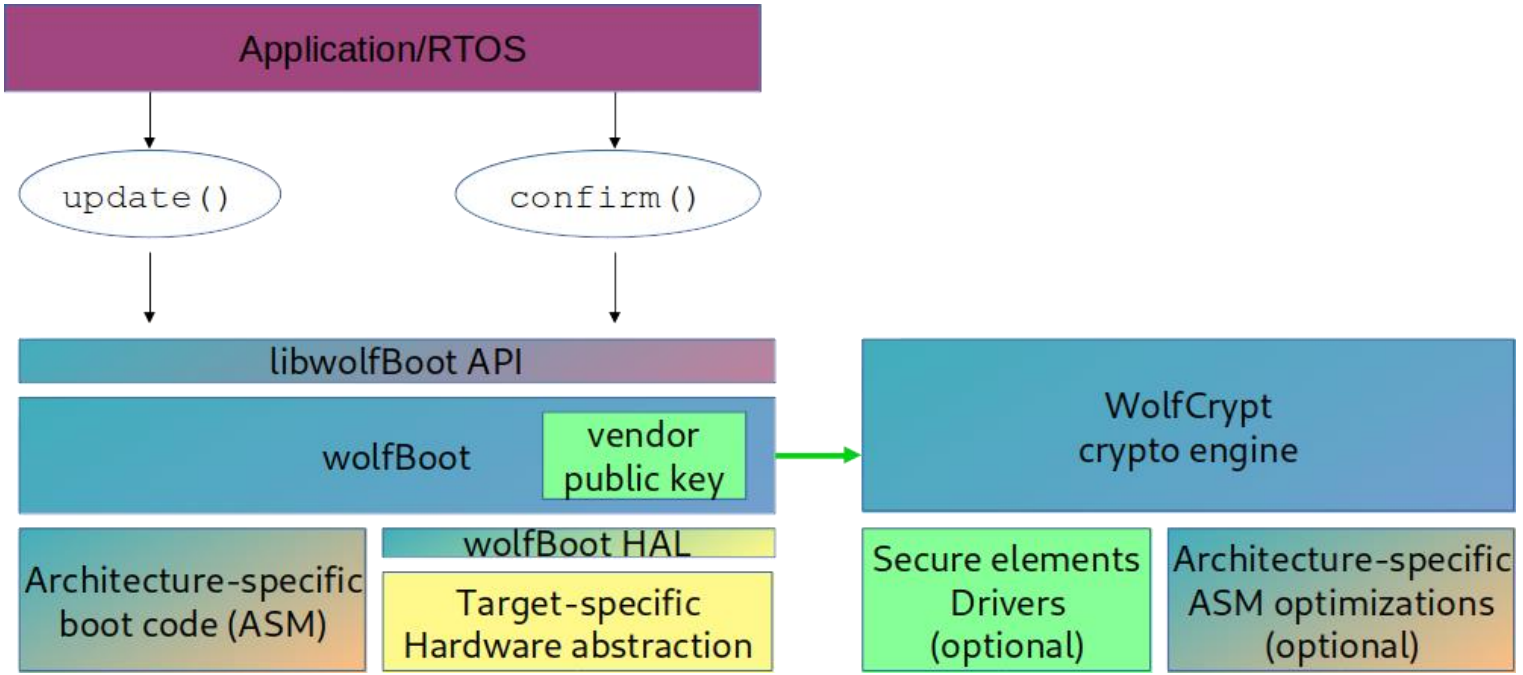
wolfBoot: Features

- Dual-bank update with fallback in case of failure
- Protection against downgrade attacks
- Interruptible (power fail-safe) software swap operation
- Support for external SPI flash memories
- Self-update (bootloader update)

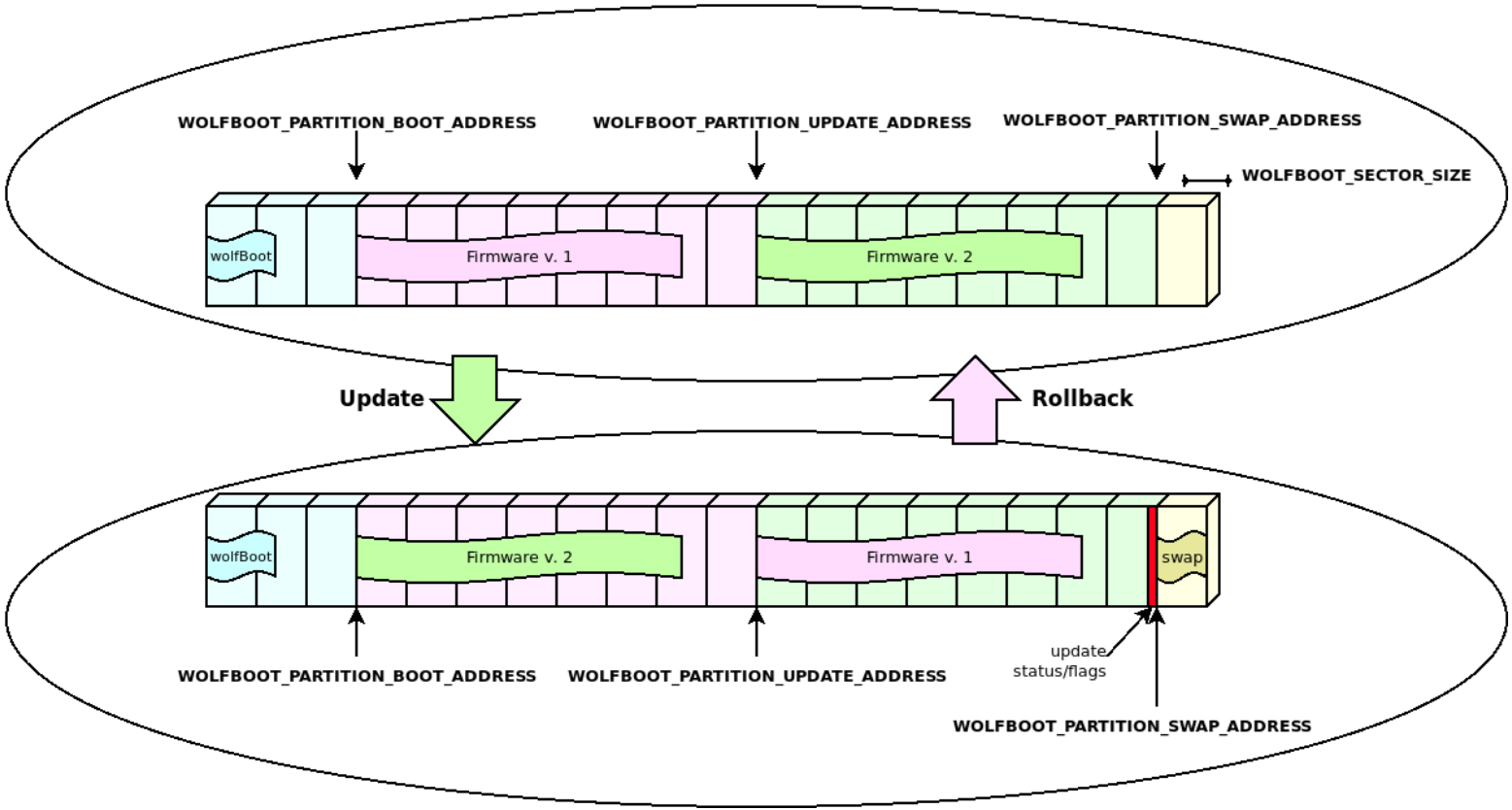
Platform specific support available:

- Secure hardware elements to assist signature verification
- Chain-load platform bootloaders to access advanced chip features
- Hardware-assisted bank swap

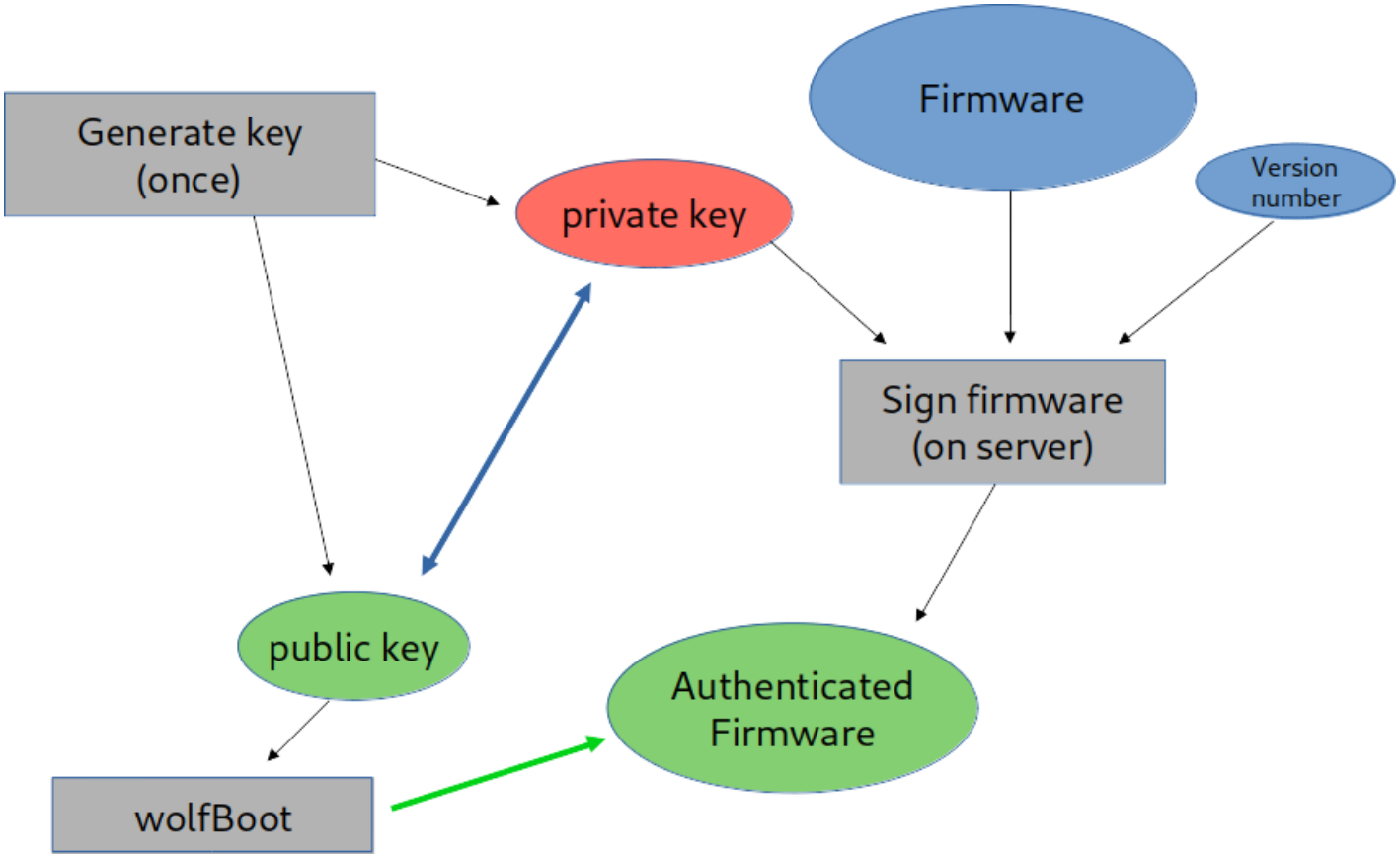
wolfBoot: Software Components



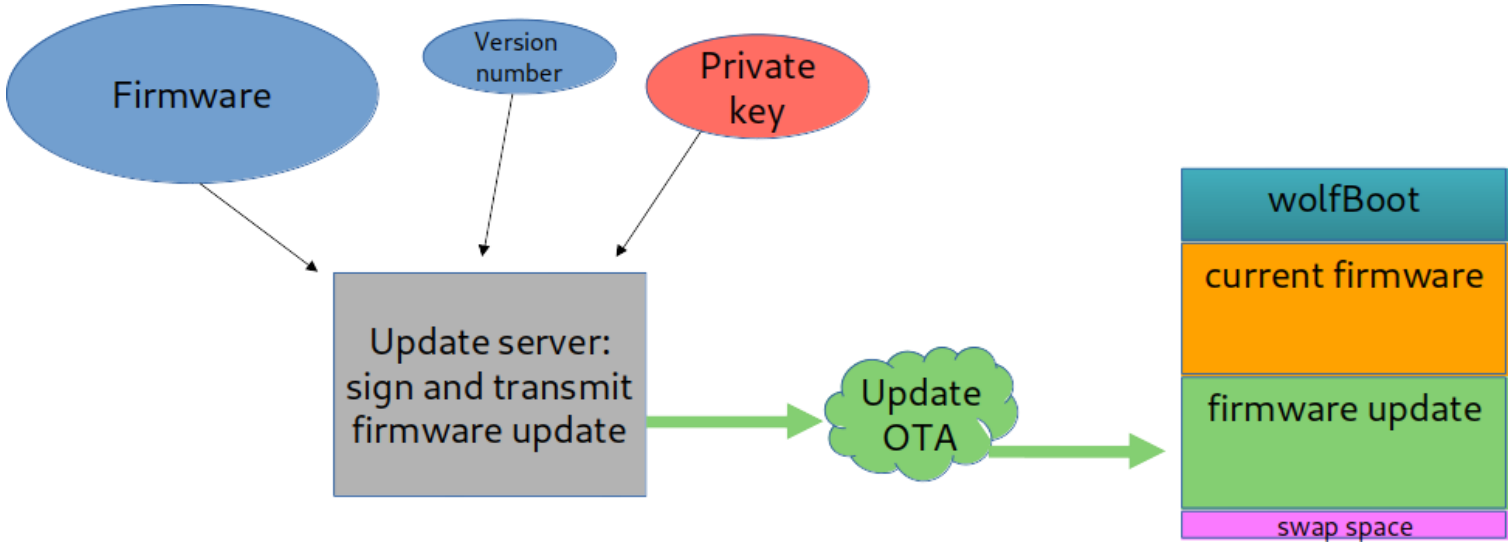
wolfBoot: Update Strategy



wolfBoot: Key Management



wolfBoot: Firmware Update



wolfBoot on RISC-V

Design & Implementation

wolfBoot on RISC-V

- Target: SiFive “HiFive1” (FE310)
  - E31 32-bit Core, 16KB RAM, 4MB QSPI Flash
- Boot Sequence
  - Reset Vector is OTP (0x1004) instructions
  - Based on MSEL pins can boot (01=QSPI0, 10=OTP 0x20000)
- Stock bootloader
  - Programmed to start of flash (0x20000000) and is 64KB
  - Allows board recovery using reset “double tap”
    - Checks reset button to stop application startup

wolfBoot on RISC-V

- RISC-V E31 Architecture specific code:
  - Relocate interrupt table
    - “mtvec” instruction to set “Interrupt Vector Base Address”
  - do_boot() function
    - Setup jump to application “jr” instruction
  - arch_reboot() function
    - Reset the microcontroller using watchdog
- FE310/Hifive1 specific code:
  - Clock PLL configuration: Use faster clock
  - eSPI driver with support for HW/SW mode

wolfBoot on RISC-V: Making it work

`do_boot()` function: Architecture specific function called after firmware image is authenticated to boot application

1. Sets new interrupt base to beginning of staged firmware (“mtvec” instr)
2. Jump to the location of the application’s main (Reset Vector)
3. Stack Pointer (SP) is updated by the application at startup

eSPI flash driver: Platform specific driver for accessing the external QSPI Flash

- During flash erase/write the function must be run from RAM
- Introduced a new compile-time flag “RAM_CODE=1” that relocates flash functions to RAM

wolfBoot on RISC-V: Performance

- Bootloader code size in flash

ECC 256-Bit, SHA256	15.0KB
ED25519, SHA512	12.5KB

- Performance at 320 MHz and SPI Dual Mode I/O at 50MHz:

ECC 256-Bit Verify	0.937 ops/sec (1.067 sec)
ED25519 Verify	0.399 ops/sec (2.506 sec)
SHA-256	512.2 KB/sec
SHA-512	270.5 KB/sec

wolfBoot Roadmap: What's to come

- RISC-V assembly optimization to reduce boot time to under 100ms
- MISRA C Version for automotive; in process now
- DO-178 certification for aviation; in process now
- FIPS 140 is available now!
- Further integration with Secure Elements / HSM's and Trusted Execution Environments.
- Operating systems; currently support or plan to support:
  - FreeRTOS, MQX, VxWorks, Zephyr, Contiki, Riot, ThreadX, Micrium
- Further Integration with new silicon as it rolls out
- Further Integration with OTA firmware updates using TLS and MQTT

wolfBoot on RISC-V: Wishlist

- **Nice-to-have** features on microcontrollers designed to implement secure updates:
  - Hardware assisted dual-bank mapping with a register to ‘swap’ the address of the two partitions

    E.g. partitions ‘A’ -> 0x2000 0000 ‘B’ -> 0x2100 0000,

    set swap bit-> ‘A’ -> 0x2100 0000 ‘B’ -> 0x2000 0000
  - On-the-fly symmetric block decryption and XIP (Execute In Place), via integrated secure elements, from memory mapped SPI memory

Questions?

Thank you!

facts@wolfssl.com
www.wolfssl.com