Chrome OS Firmware

December 2017
Duncan Laurie

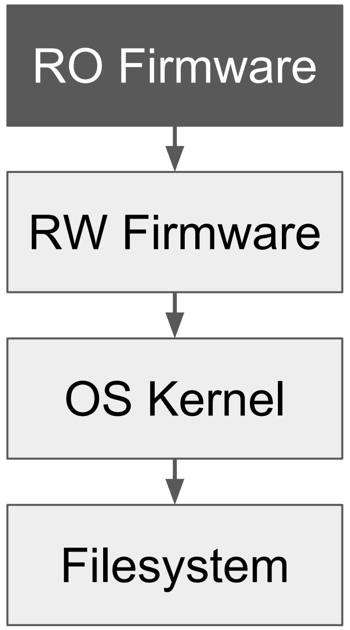
Verified Boot

Verified Boot Objectives

- Provide a secure computing platform for users
- Be as fast as possible while doing it
- Provide hardware level recovery from remote attacks
- Allow users to turn off verification
- Does not protect against local attack that requires opening device

Verified Boot

- Root of trust is Read-Only SPI flash
  - Enforced by hardware write protection
  - Contains first executed code (bootblock)
  - Public keys to verify other stages
- RO firmware verifies RW firmware
- RW firmware verifies OS kernel
- OS kernel verifies root filesystem



Verified Mode

- Boots only Google-signed Chrome OS images
- Performs a full verification of firmware and kernel
- Uses RO + RW firmware
- Boot in recovery mode if verification fails
    - RW firmware corrupt
    - Block device corrupt
    - Hardware failure

Recovery Mode

- Used to get a Chrome OS device back to a trusted state
- RO firmware can only boot signed USB image
  - Must be signed by Google with Recovery Key
  - Different from normal boot key
- Can be initiated by the system (untrusted)
  - Due to verification or hardware fault
- Can be initiated by the user with physical presence (trusted)
  - Boot recovery image from USB
  - Switch to developer mode

Developer Mode

- Turn Kernel verification off without voiding warranty
- Can be disabled with enterprise policy
- User is warned at boot that verification is turned off
- Requires physical presence (via recovery mode) to enable
- Device state (including TPM owner) is cleared when transitioning
- Provides increased access to the system
  - Root shell
  - Boot images signed with different keys
  - Set flags to boot from USB or in legacy mode

Legacy Mode

- Unsupported method for booting alternate OS
- Can boot any payload from flash
  - Currently include SeaBIOS for legacy CSM
  - Moving to TianoCore
- Must be enabled after transition to developer mode

Complete Device Ownership

- Developers or those who do not want to use Chrome OS
- Open device case to disable write protect
  - Typically voids warranty
- Set RO flags that alter boot behavior
  - Short developer screen timeout
  - Force enable of developer or legacy mode so it cannot be disabled at boot
- Flash complete custom firmware image
  - Download and build from coreboot.org

Verified Boot and TPM

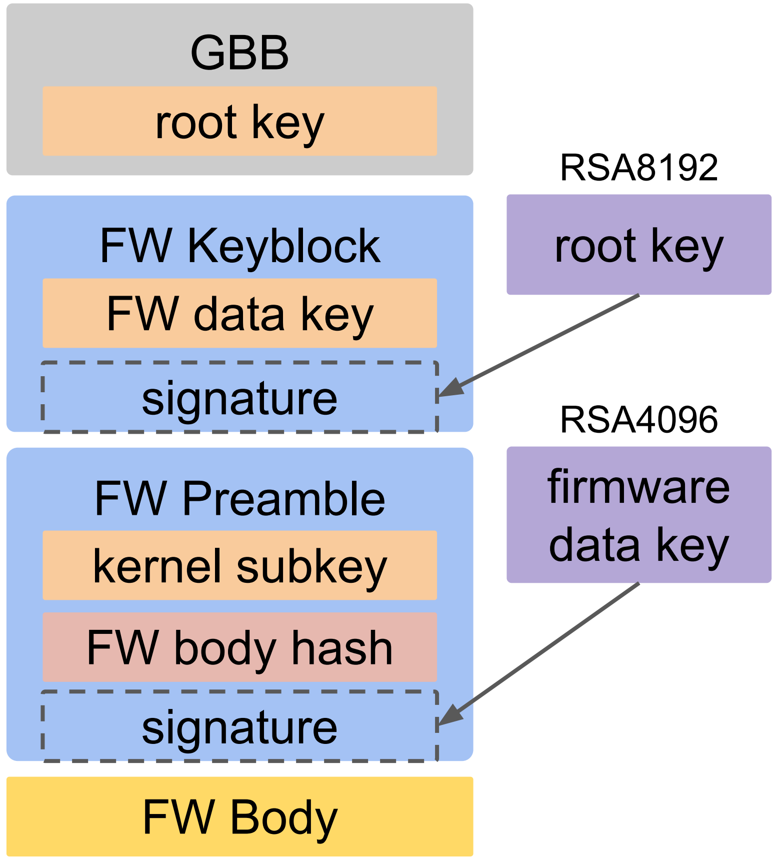
- Must contain Google certificate to identify Chrome OS device
  - This certificate cannot be made available to other OS
- Firmware does not use TPM for cryptographic functions
  - Preventing key rollback attacks
- Firmware and Kernel key versions in NVRAM
  - Store boot mode state (developer, recovery)
- Platform Configuration Registers
  - TPM is locked by firmware, except in recovery mode
  - Physical Presence is asserted and locked
- TPM hardware
  - New Chrome OS devices use Google H1 security chip (TPM 2.0 compatible)
  - Drivers provided by coreboot/Depthcharge
  - Must be available before memory is up

Firmware Verification

Read-only firmware looks in the GBB, finds the **root key** in read-only flash.

The root key validates the RW firmware's keyblock, which contains the **firmware data key**.

The firmware data key validates the firmware preamble (which contains the **kernel subkey**) and the read/write firmware body.

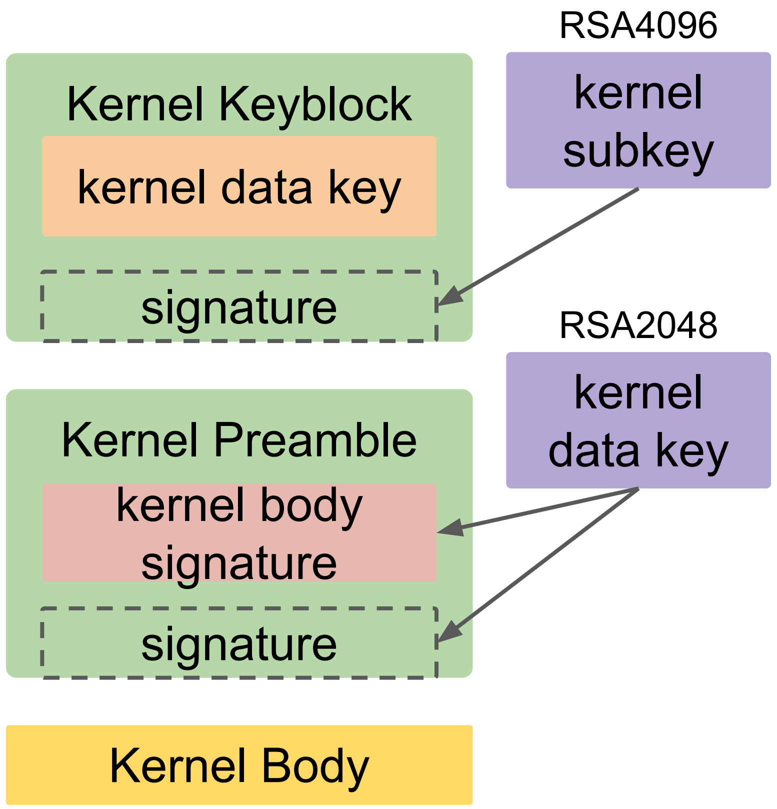


Kernel Verification

RW firmware uses the **kernel subkey** to verify the kernel's keyblock, which contains the kernel data key.

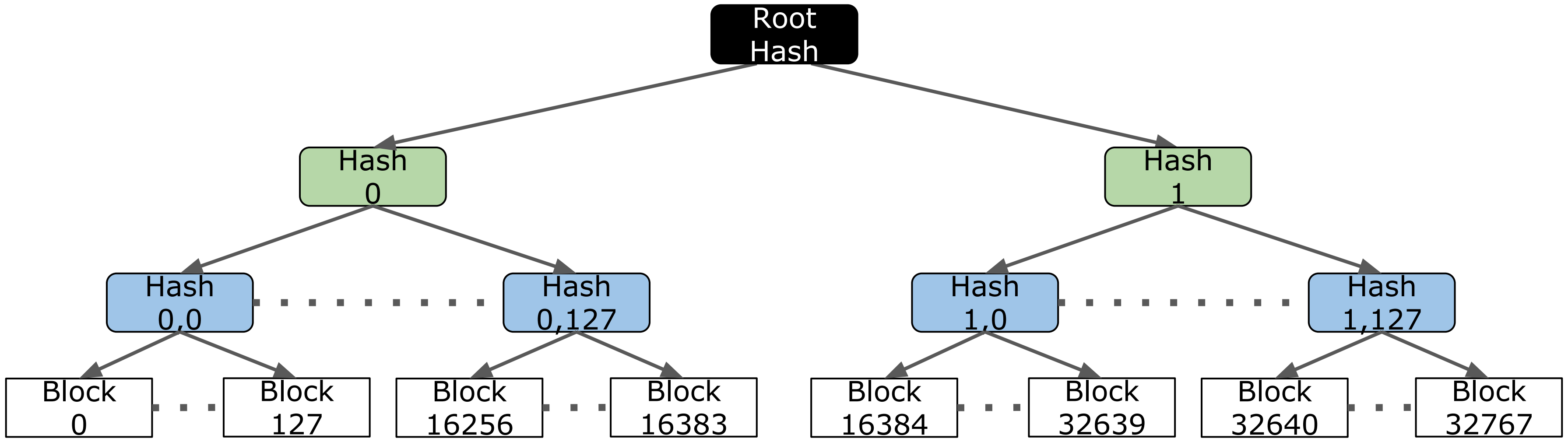
The **kernel data key** verifies the kernel preamble and the kernel body.

The kernel image contains a hash, used to verify the bundle of disk block hashes.



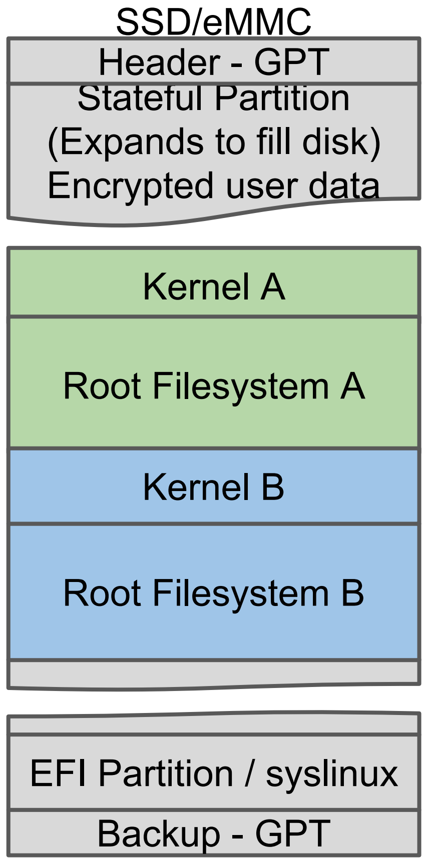
Filesystem Verification

- Root filesystem is a Read-Only image
  - Hash each block in the image and put into a tree
- On boot, root hash is specified on kernel command line and signed
- Subsequent blocks are read, hashed, and stored in page cache



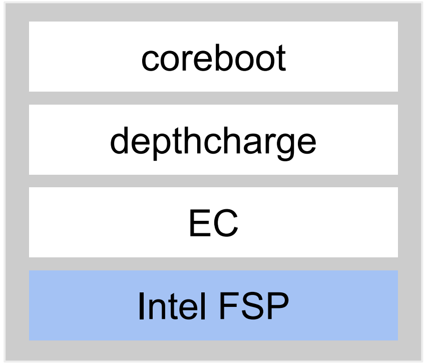
Verified Boot Disk Layout

- Two root filesystems and two kernel partitions
- Kernel A pairs with file system A
- Kernel B pairs with file system B
- One pair is active and the other is backup
- The active partition is read-only
- The inactive partition can be updated
- EFI partition contains grub and syslinux
    - Provided for booting on non-Chrome devices



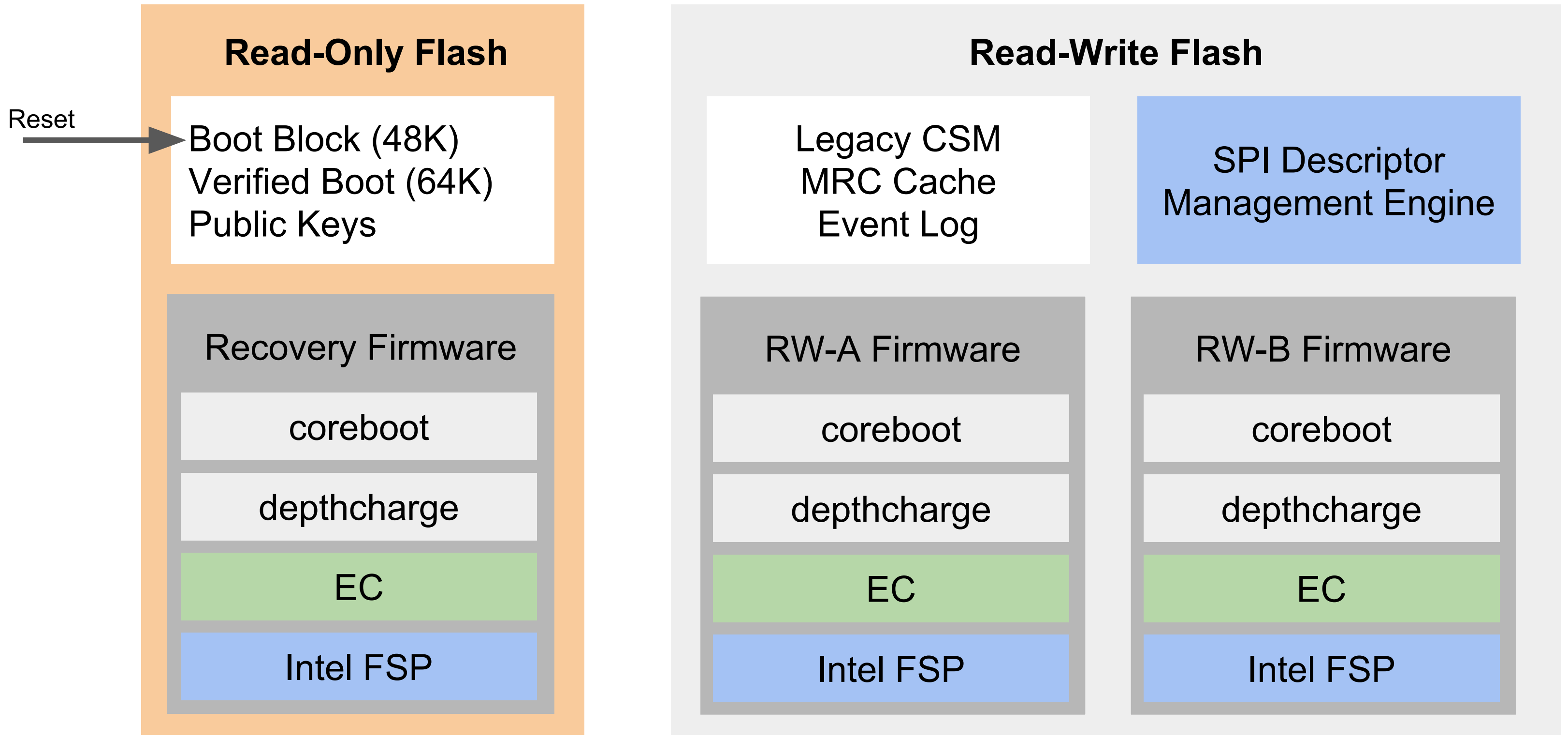
Firmware Architecture

Chrome OS Intel Firmware Architecture

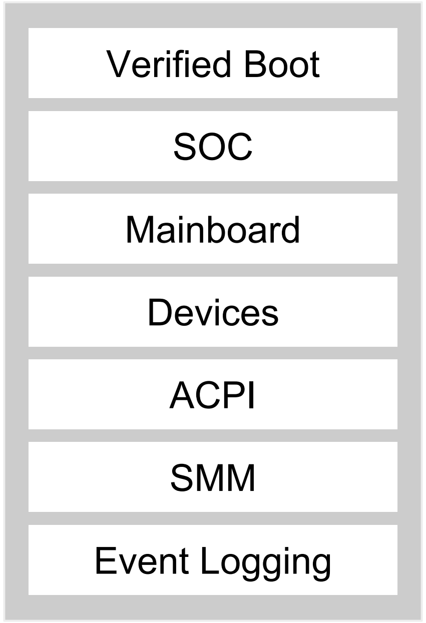


- 3 copies of the firmware in flash
  - 1 in RO flash for recovery mode
  - 2 in RW flash for verified A / B boot
- Each firmware component is independent
  - Flexibility in development
  - Distinct boundaries for different code licenses
  - APIs for passing data

Firmware Layout

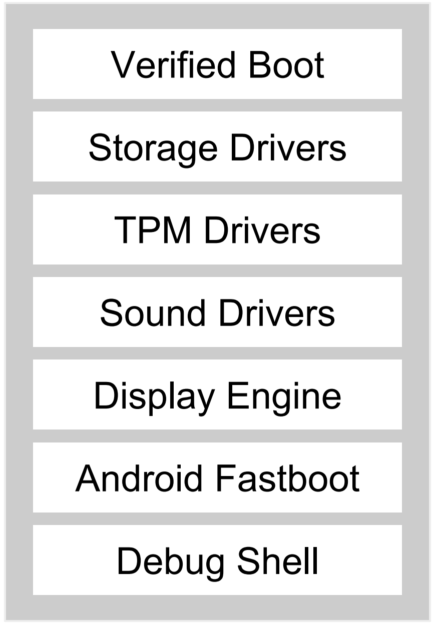


coreboot



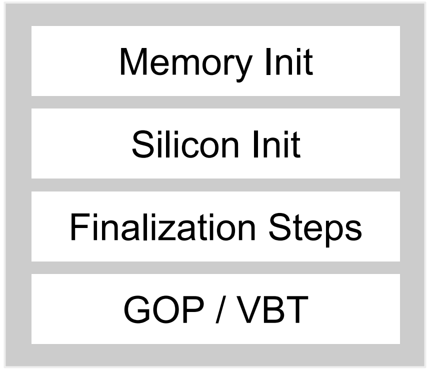
- GPLv2 License
- Designed for code sharing and re-use
- Support all architectures, SOCs, boards in one tree
- Intel SOC support is co-developed by Google and Intel
- Board support is based on reference board model
- Verified Boot does firmware selection (A/B/recovery)
- SMM use is intentionally *very* minimal
- Event logging is extensive

Depthcharge



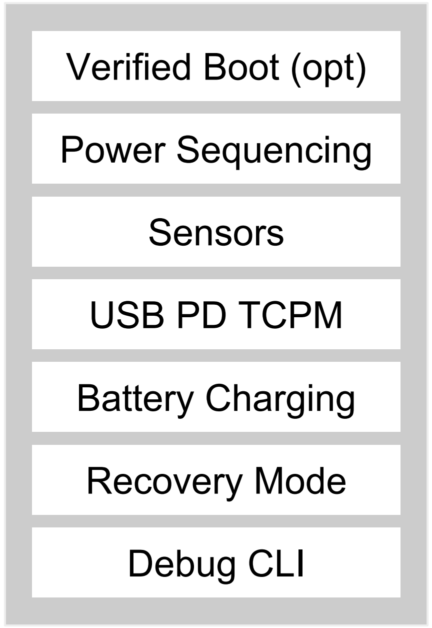
- GPLv2 License
- Chrome OS payload for coreboot
- Storage drivers for SATA, eMMC, NVMe, USB
- Sound drivers for beep (security/accessibility requirement)
- Display engine for drawing localized firmware screens
- Verified Boot does kernel selection based on GPT flags
- Handles Chrome OS boot modes
  - Verified Mode
  - Recovery Mode
  - Developer Mode

Intel Firmware Support Package



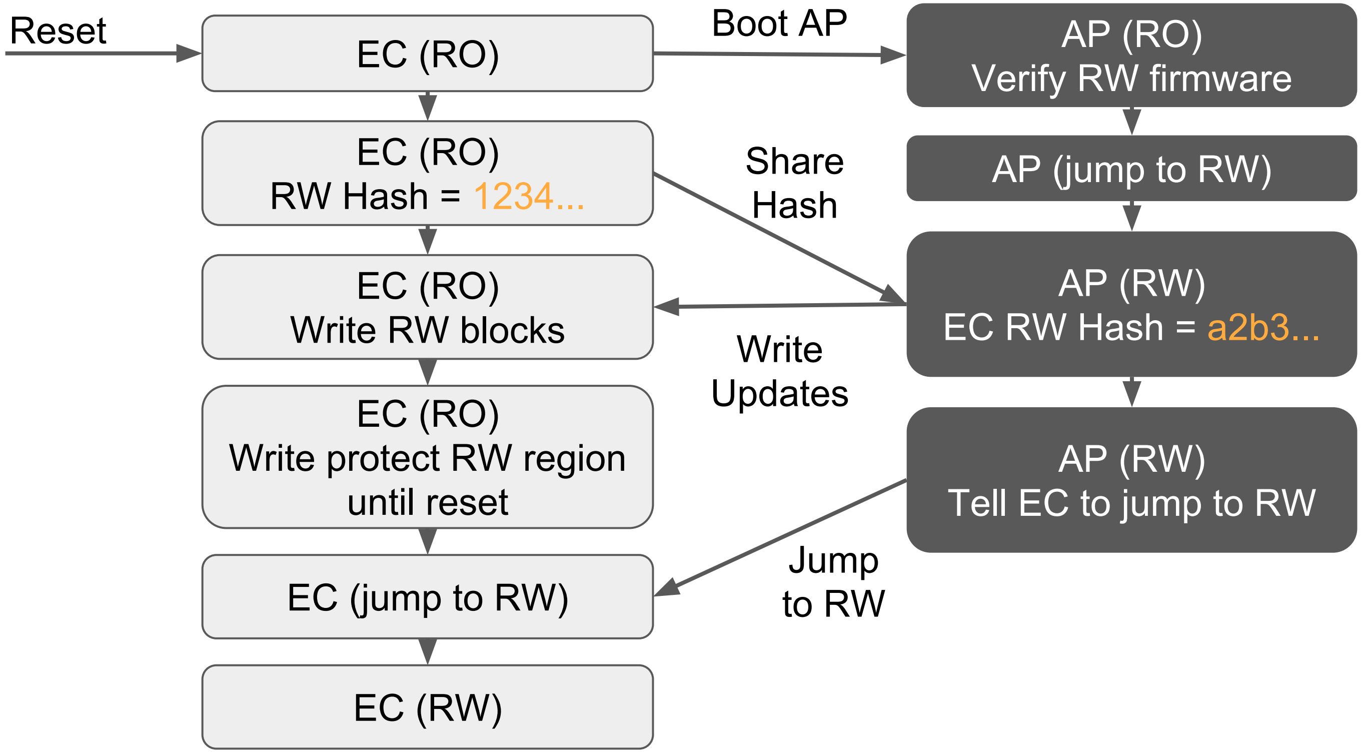
- Intel Proprietary License
- FSP Spec designed by Intel and Google
- Provided by Intel in source and binary form
- Responsible for most SOC setup
  - Memory, CPU, PCH, GPU, ME, etc
- Configuration via UPD
  - Turned into “Policy” settings for reference code

Embedded Controller



- BSD License
- Supports RO/RW firmware model
  - Both copies are the same
  - Can “jump” between images
- Verified Boot Software Sync for updates
  - Compare hash between EC and update data
  - Push down new RW update if needed
  - Jump to new RW image before loading OS
- Can also do Verified Boot by itself
  - With more impact to boot time

EC Software Sync



Management Engine

- Intel Management Engine security is inadequate
  - Requires RW access to flash and host memory
  - Reduced control over the platform
  - Increased complexity in host firmware
- Out of band access to the system is a crutch for not having OS control
  - Vertical integration of Chrome OS makes this unnecessary
- Closed source firmware with numerous security issues
  - Running unsigned code on Intel Management Engine (2017)
  - ThinkPwn UEFI bug affects multiple vendors (2016)
  - Hacking Team releases UEFI rootkit (2015)
  - New attack methods to defeat Secure Boot (2014)

Porting

Intel SOC Support in coreboot

- Leverage common code for reduced porting effort
- Provide wrapper for platform-specific FSP configuration
- SOC device setup that is not handled by FSP
- ACPI code for SOC devices
- GPIO community description
- Intel provided binaries for the SOC
    - GOP and VBT binaries for video initialization
    - NHLT binaries for OS sound drivers
    - Microcode binaries for the CPU

Intel SOC Common Code

- Significant code sharing across SOC
- Simplify SOC porting effort
- Co-developed by Intel and Google
- Reduced code for SOC support
    - Skylake+Kabylake: 8000 lines
    - Apollolake: 4000 lines
    - Cannonlake: 4000 lines

- System Agent
- CPU
- UART drivers
- GPIO setup and control
- I2C controller init and drivers
- SPI controller init and drivers
- ACPI code
- SMM
- SGX

Mainboard Support in coreboot

- Abstracted device tree with configuration data
- Builds on SOC support and common code
- Very little C or ACPI code for a mainboard
    - Flexible enough to support it for debug or workarounds
- Many configuration items are common no matter what firmware base is used
    - Intel reference code policy settings (aka FSP UPD)
    - On-board memory configuration
    - GPIO configuration
    - On-board devices

Mainboard Porting: Baseboard & Variants

- Reference board model applied to firmware
- Baseboard
  - Co-developed by Google and Intel to support Chrome OS ecosystem
- Variant
  - Developed by partners for a shipping product
  - Support from Google and Intel
- Variant is applied as an overlay on top of baseboard
  - Inherit by default, but can customize
- Variant is very low effort porting from reference board

Mainboard Porting Example: Pixelbook

- 500 lines of C code
  - ½ is for GPIO configuration described as a C structure
- 100 lines of ACPI code (for the DPTF profile)
- 400 lines of Device Tree
  - Board specific FSP settings
  - Voltage regulator configuration
  - PCIe root port configuration
  - USB port configuration
  - On-board device description
- SPD for onboard LPDDR3 memory

Mainboard Porting Example: Touchpad

```
register “i2c[2].voltage” = “I2C_VOLTAGE_1V8”              # 1.8V bus
register “i2c[2].speed” = “I2C_SPEED_FAST”                 # 400 KHz
register “i2c[2].rise_time_ns” = “112”                     # Measured rise time is 112ns
register “i2c[2].fall_time_ns” = “34”                      # Measured fall time is 34ns

device pci 15.2 on                                         # PCH I2C bus 2
    chip drivers/i2c/hid                                   # I2C-HID device on this bus
        register “irq” = “ACPI_IRQ_LEVEL_LOW(GPP_B3_IRQ)”  # IRQ is routed to GPP_B3
        register “wake” = “GPE0_DW0_15”                    # Wake from GPE0_DW0 block pin 15
        register "hid_desc_reg_offset" = "1"               # HID Descriptor register offset 1
        device i2c 49 on end                               # I2C slave address 0x49
    end
end
```