Verified Boot and Free Software: Reconciling Freedom and Security

Paul Kocialkowski
contact@paulk.fr

Monday July 4th 2016

Introducing Verified Boot and Related Issues

Introducing Verified Boot and Related Issues

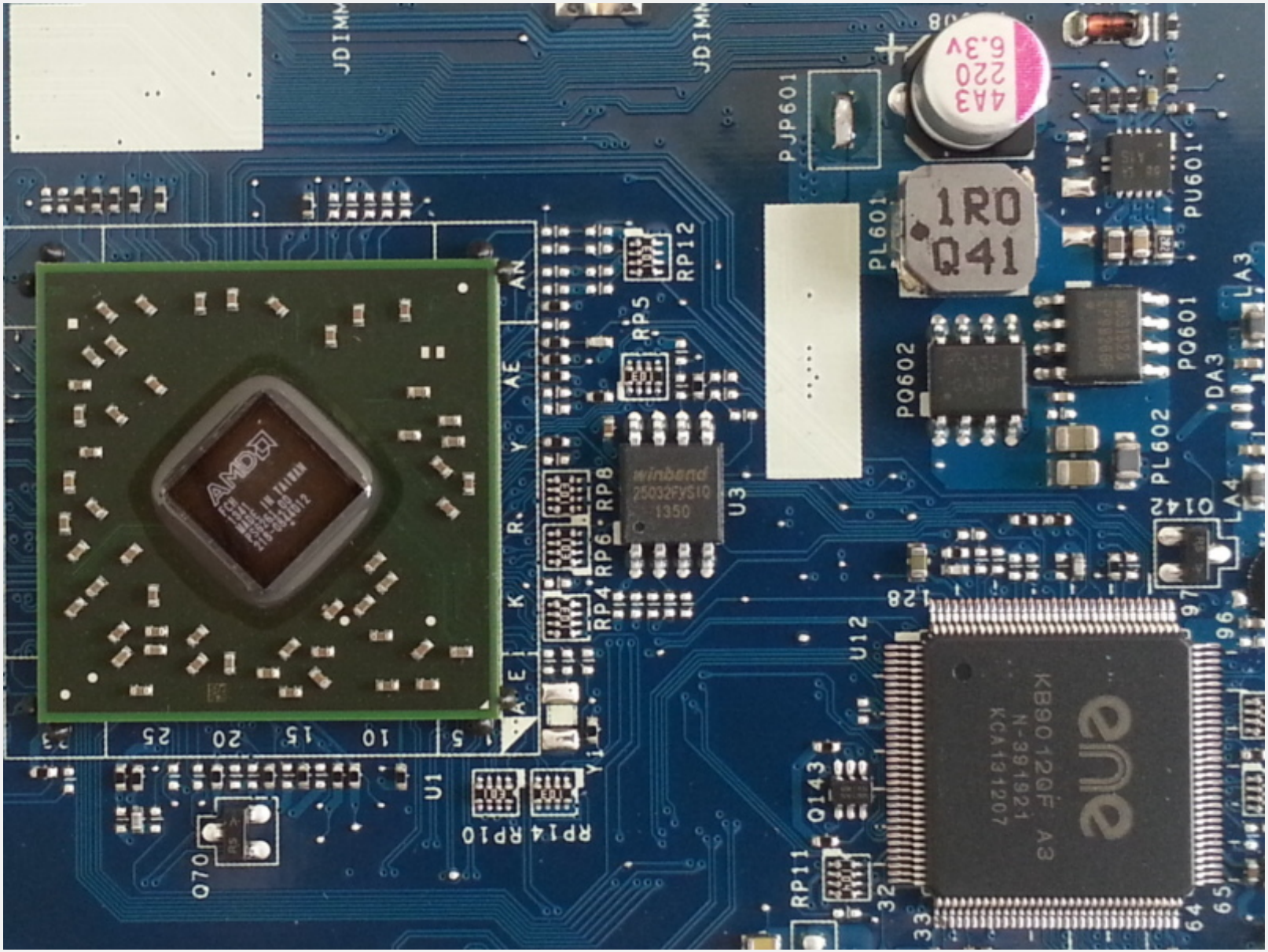
Bootup Process

BIOS History

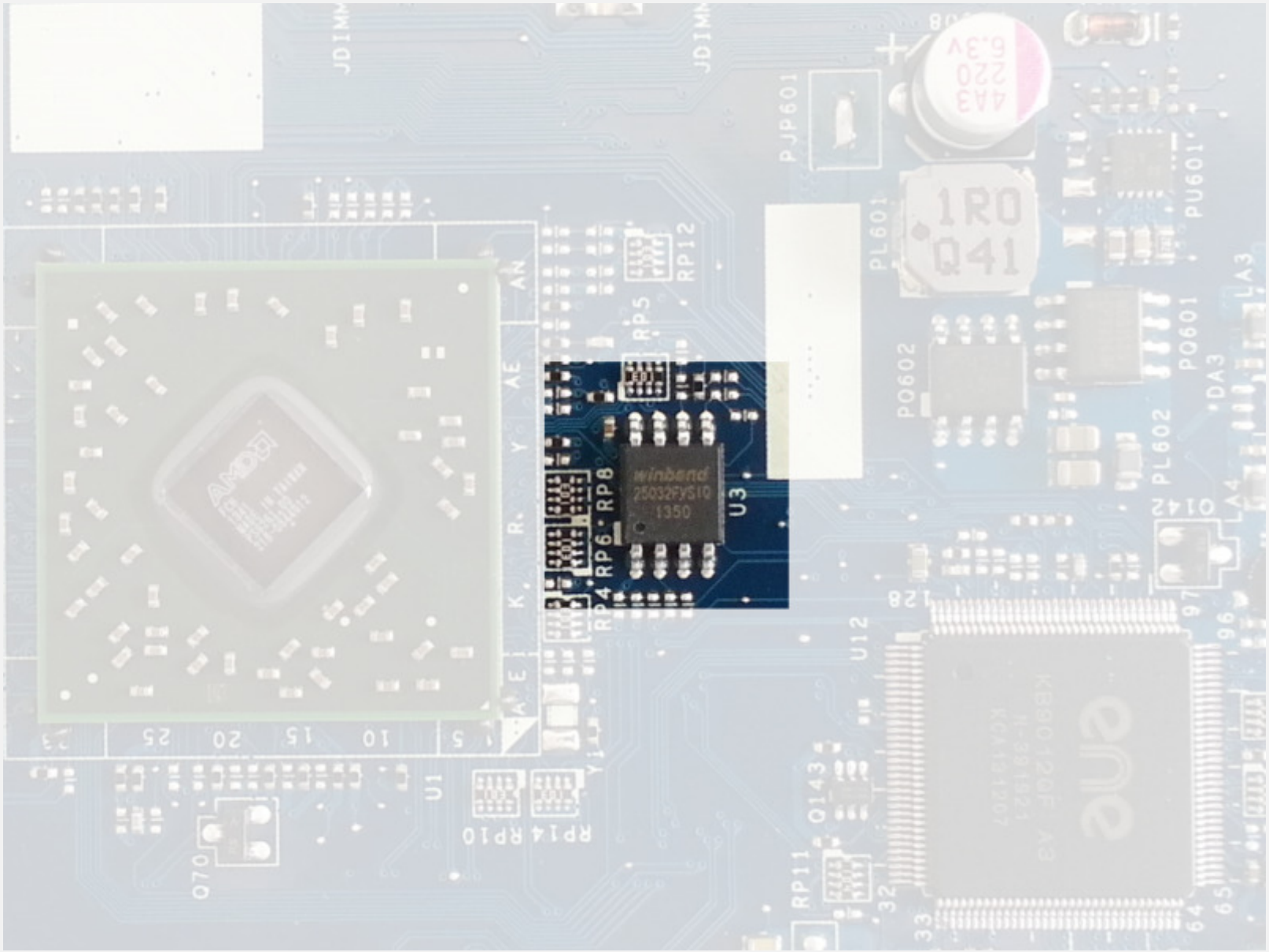
Basic Input/Output System:

- 1980s:
  - Basic hardware initialization
  - Operating system load
  - BIOS interrupt calls (used by CP/M, DOS)
  - Purpose: hardware abstraction
  - Read-only memory
- 1990s:
  - Increasing hardware complexity
  - Drivers in operating systems, initialization only
  - Read/write memory (updates)
- 2000s:
  - Run-time services (SMM/SMI, ACPI)
  - Unified Extensible Firmware Interface (UEFI)
  - Back to hardware abstraction

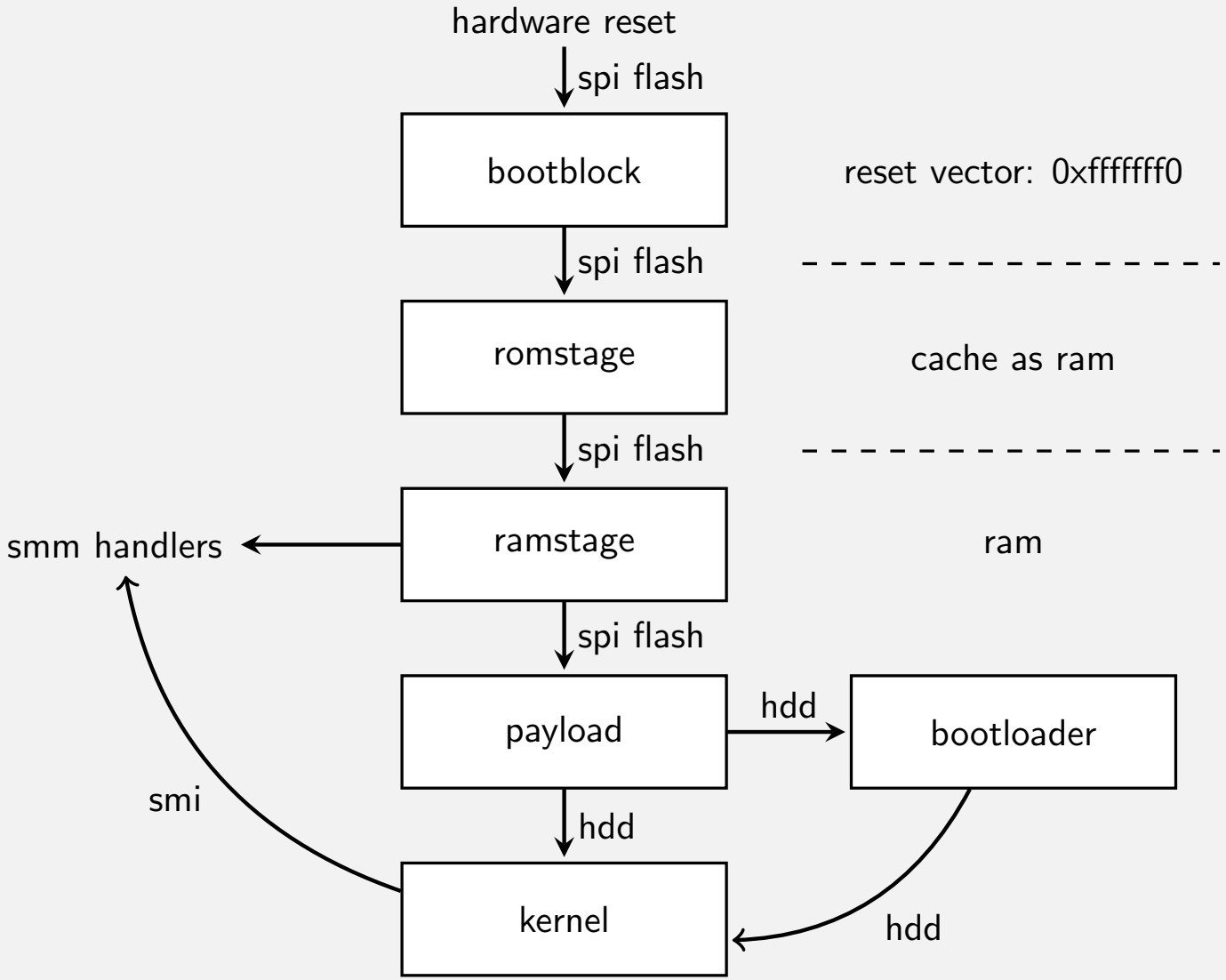
SPI Flash?


JDIMM
JDIMM
PJP601
4A3
220
6.3V
PL601
1R0
Q41
PU601
PO602
PO601
PL602
DA3
LA3
LA4
Q142
RP12
RP5
RP8
RP6
RP4
U3
winbond
25Q32FVSIQ
1350
U12
128
97
96
ene
KB9012QF A3
N-391921
KCA131207
Q143
RP11
32
33
64
65
U1
Q70
RP14
RP10
AN
AE
Y
R
K
E
A
Y1
5
10
15
20
25

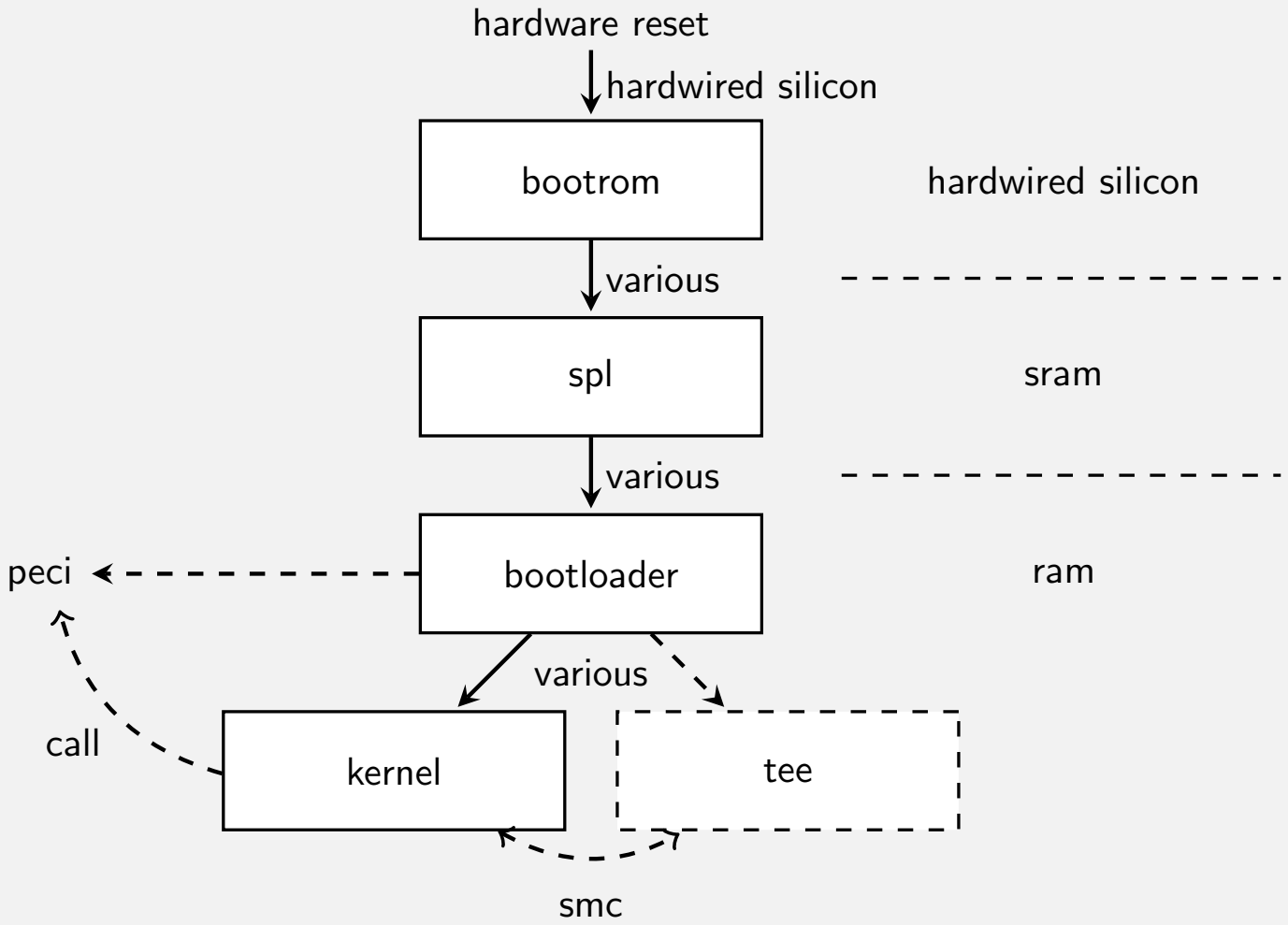
SPI Flash


winbond
25Q32FVSIQ
1350
U3
RP4
RP6
RP8
RP5
RP12
PJP601
PL601
PU601
PO602
PO601
PL602
U12
U1
RP14
RP10
RP11
Q143
Q142
Q70
1RO
Q41
4A3
220
6.3V

(Traditional) x86 Bootup Process



ARM Bootup Process



Trusted Execution Environment

Need for trusted run-time software:

- Operating system is flawed
- Privileged operations, hardware access
- Sensitive operations (privacy/security)

Implementing a trusted environment:

- Independent or setup by bootup software
- Cooperation with the platform (e.g. **TrustZone**)
- Privileged mode, interfacing (e.g. SMC)

Introducing Verified Boot and Related Issues

Software Freedom

Freedom and Privacy/Security Considerations and Issues

Bootup software considerations:

- Precedence over the system
- *Runs during the system lifetime*
- *Loads the TEE*
- Great control, abilities and user data access
- Hardware initialization knowledge

Associated issues:

- Trust and control:
  - Audit (weaknesses, backdoors)
  - Bug fix (delays, EOL)
  - User modification
- Restrictions
- Access to knowledge

Basic Freedoms

Guarantees: basic freedoms

0. Run for any purpose
1. Study and modify
2. Redistribution
3. Redistribution of modifications

Free software is a hard requirement!

Free Bootup Software

Bootrom (ARM):

- Read-only: **hardwired**
- Always **non-free** (hardware design)

Free bootup software projects:

- Coreboot
- U-Boot, Barebox
- Libreboot

Usual non-free components:

- **x86**: Option ROM/VGA BIOS, CPU microcodes
- **Intel x86**: FSP, MRC, ME
- **AMD x86**: IMC, SMU, PSP
- Various firmwares (xHCI, ethernet, . . .)

Introducing Verified Boot and Related Issues

Bootup Software Verification

Early Software Verification

Security approaches distinction:

- Verified boot: to boot or not to boot
- Measured boot: state indication

Verified boot rationale:

- Read/write bootup software
- Attack surface, compromisation
- Chain of trust up to the system, handlers

Design implications:

- Early bootup software must be trusted
- Next stage validation: signatures
- Read-only signatures

Early Attempts at Integrity Preservation

Pin-driven write-protect:

- Easy to find out
- Flashrom board enables

flashrom/board_enable.c:

```
static int intel_piix4_gpo27_lower(void)
{
    return intel_piix4_gpo_set(27, 0);
}
```

Platform-based approaches:

- Platform SPI access/write disable
- Protected Ranges (PRx)

Pitfalls:

- Privileged access (SMM)
- Internal controller access (ME, IMC)
- External access
- Platform-specific logic

UEFI and Secure Boot

Secure boot implementations:

- Bootloader or kernel verification
- Ships with verification keys
- Assumes it's not compromised

User control:

- Secure boot disable?
- Replacing keys?

(Actually) Verified Boot

Strong root of trust:

- Keys in OTP memory
- Bootrom enforcing verification

Motivations:

- Multiple read/write storage
- Reliable root of trust

Implementations:

- ARM platforms
- Intel Bootguard
- Intel TXT ACM (dynamic root of trust)

Introducing Verified Boot and Related Issues

Software Freedom Issues

Incompatibility with Free Software

Issues for freedom:

- Free bootup software is **impossible**!
- Free TEE software is **impossible**!
- *Free kernel is* **impossible***!*

Implications for privacy/security:

- No control, no choice for trust
- (Un)intentional weaknesses
- Ability to compromise the system, exfiltrate data:
  - At boot time
  - At run time: TEE, SMM/SMI, ACPI, PECI

Neither freedom, nor privacy/security :(

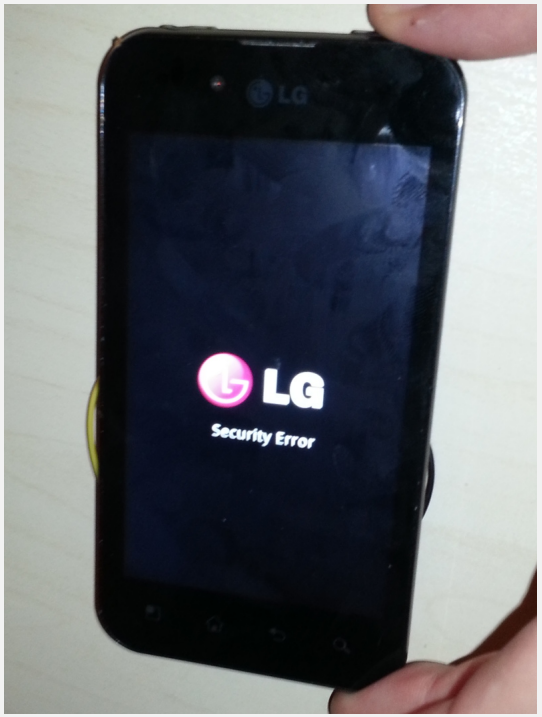
(Barely) Undercover Motivations

Usual scenario (Android):

- Verified SPL and bootloader
- Verified TEE
- Chain of trust break

Story time:

- LG Optimus Black
- Google Pixel C



Rationale? Anyone? It's pronounced **DRM**.

Tivoization and Licenses

Tivoization:

- Free, copyleft bootup software
- Modified versions release
- Signed binaries for verified boot

Case study:

- Samsung Galaxy Tab 2
- X-Loader SPL

Classification	Model	Version	Source Code	Announcement	Inquiry
Mobile Phone	GT-P5100	P5100XXDN...	GT-P5100_JB_Opensource_Update3.zip		
Mobile Phone	GT-P5100		GT-P5100_JB_Opensource_Update2.zip		
Mobile Phone	GT-P5100		GT-P5100_JB_Opensource_Update1.zip		
Mobile Phone	GT-P5100		GT-P5100_ICS_Opensource_Update1.zip		
Mobile Phone	GT-P5100		GT-P5100_ICS_Opensource.zip		

Tivoization and Licenses

Tivoization:

- Free, copyleft bootup software
- Modified versions release
- Signed binaries for verified boot

Case study:

- Samsung Galaxy Tab 2
- X-Loader SPL

x-loader/SecureBootSign.pl:

```
$RemoteServer1 = "10.254.124.52";
$RemoteServer2 = "10.90.13.36";
[...]
$handle = IO::Socket::INET->new("$RemoteServer:80") or $newerr=1;
$handle->send("GET http://$RemoteServer/SigningKey.php?model=$modelname&type=
      $runtype&filename=$input_filename&filepath=$ftp_sub_path&prefix=$prefix&
      viewtype=$viewtype&ppa=$config_filename HTTP/1.0");
```

Tivoization and Licenses

Tivoization:

- Free, copyleft bootup software
- Modified versions release
- Signed binaries for verified boot

Case study:

- Samsung Galaxy Tab 2
- X-Loader SPL

Oh, the irony!

Licenses:

- GPLv3

All About the Main CPU?

Software is everywhere:

- Main processor: Bootup, TEE, System
- Management processors: ME, IMC, SMU, BMC...
- Auxiliary processors: GPU, VPU, DSP...
- Controllers: EC, xHCI, multimedia, battery...
- Peripherals: Wi-Fi, bluetooth, GPS, webcam...

Verified boot:

- Main processor: **common** (ARM)
- Management processors: **common**
- Auxiliary processors: **increasing** (GPUs)
- Controllers: **uncommon**
- Peripherals: **uncommon**

Freedom and privacy/security everywhere?

Reconciling Freedom and Security

Reconciling Freedom and Security

General-Purpose Possibility

Coreboot, GRUB and PGP

Verified boot with free software example:

- Free bootup software (Coreboot)
- Payload with PGP verification (GRUB)
- Storage set read-only (or hidden)
- External access to storage

Platform assumptions:

- No signature verification from bootrom
- Ability to lock the storage (access/write/regions)

Possible with non-Bootguard x86 platforms!
(with Coreboot support)

SPI Flash Write-Protect

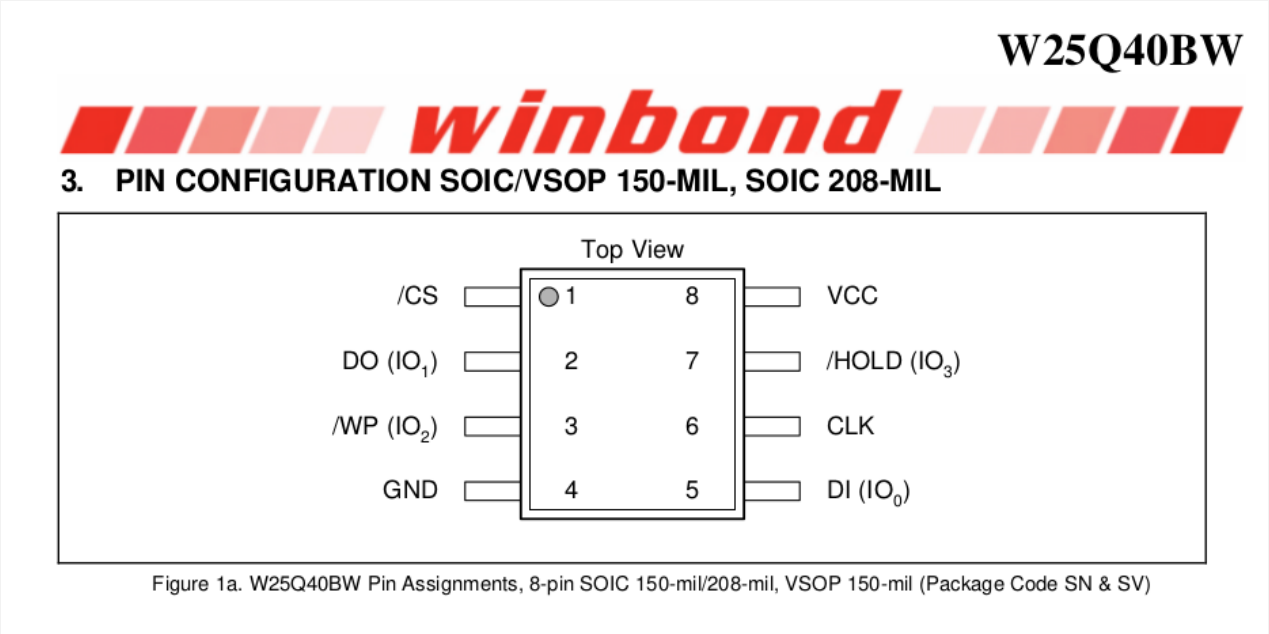


Figure 1a. W25Q40BW Pin Assignments, 8-pin SOIC 150-mil/208-mil, VSOP 150-mil (Package Code SN & SV)

Write-protect ($\overline{WP}$) pin:

- Reliable root of integrity!
- Physical switch
- Solder it to ground

Reconciling Freedom and Security

CrOS Security Model and Devices

Design Guidelines

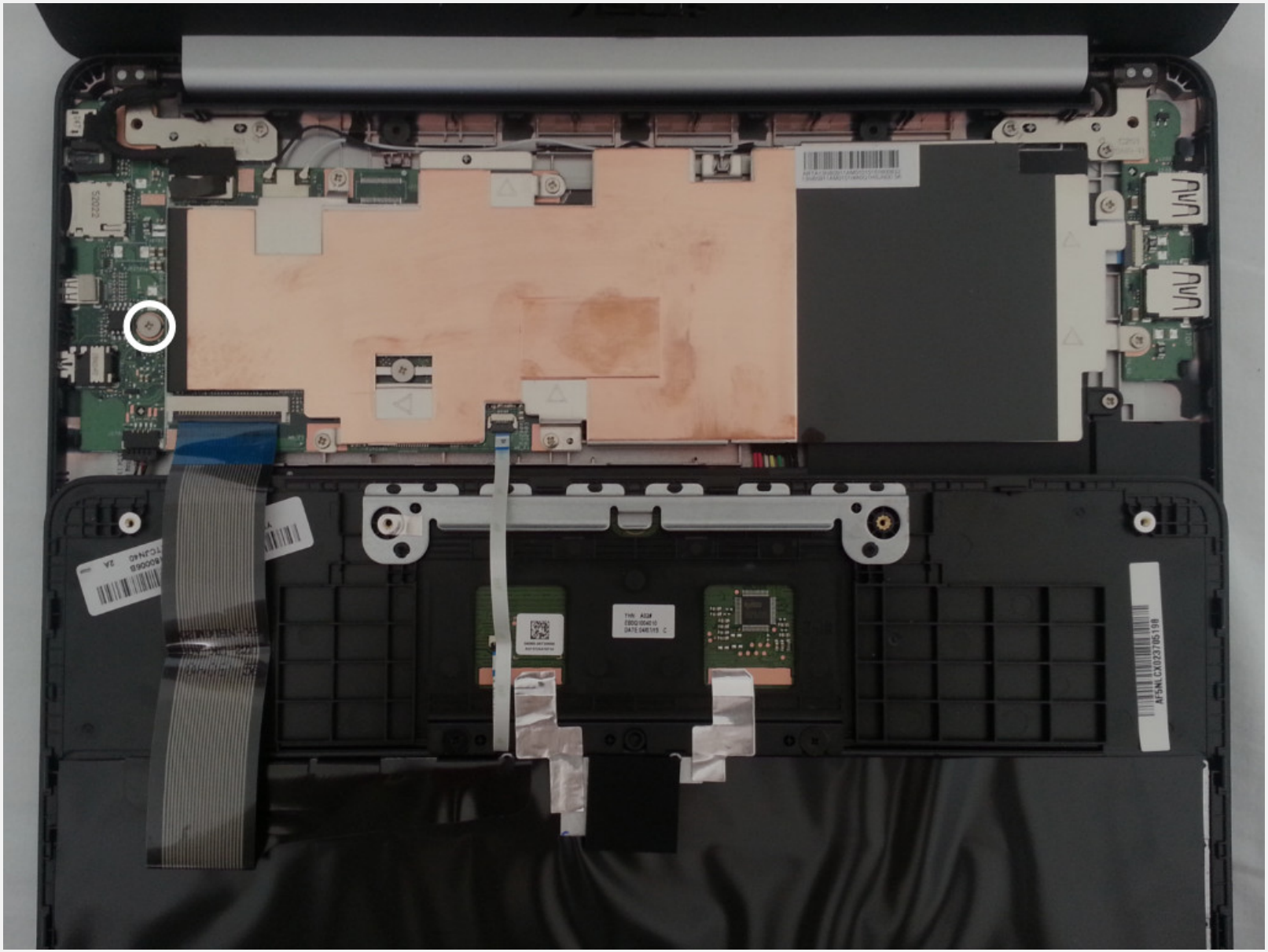
CrOS security design:

- Reliable, scalable verified boot for CPU and EC
- Does not cover external access (evil maid)
- Free software, user-friendly

Chain of trust:

- SPI flash write-protect root of trust
- The screw: write-protect switch
- RO early stages, keys and recovery
- RW (verified) next stages and kernel

The Screw



Verified Boot Software

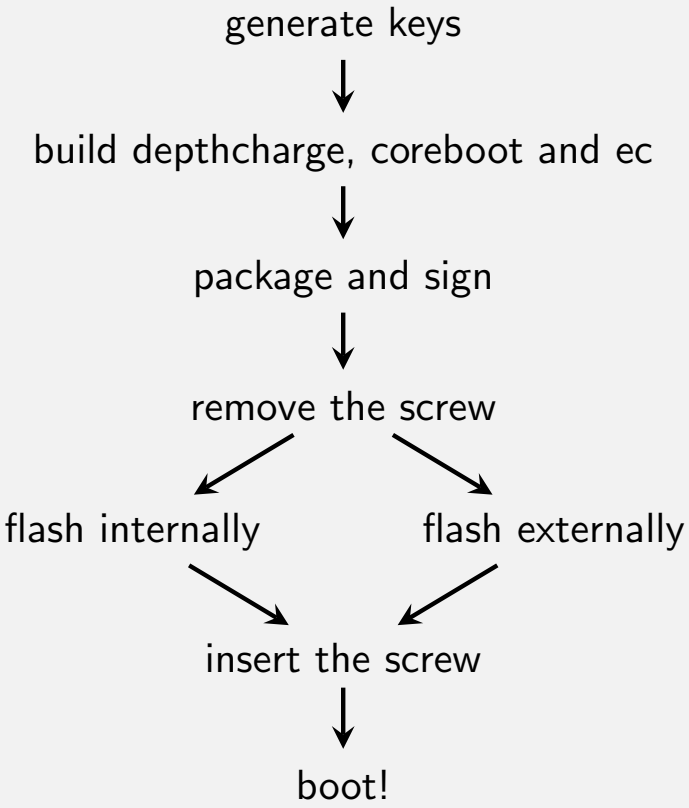
Software components:

- Bootup software: Coreboot
- Payload: Depthcharge
- Verified boot: Vboot
- EC firmware: Chrome EC

Boot modes:

- Normal mode
- Recovery mode
- Developer mode

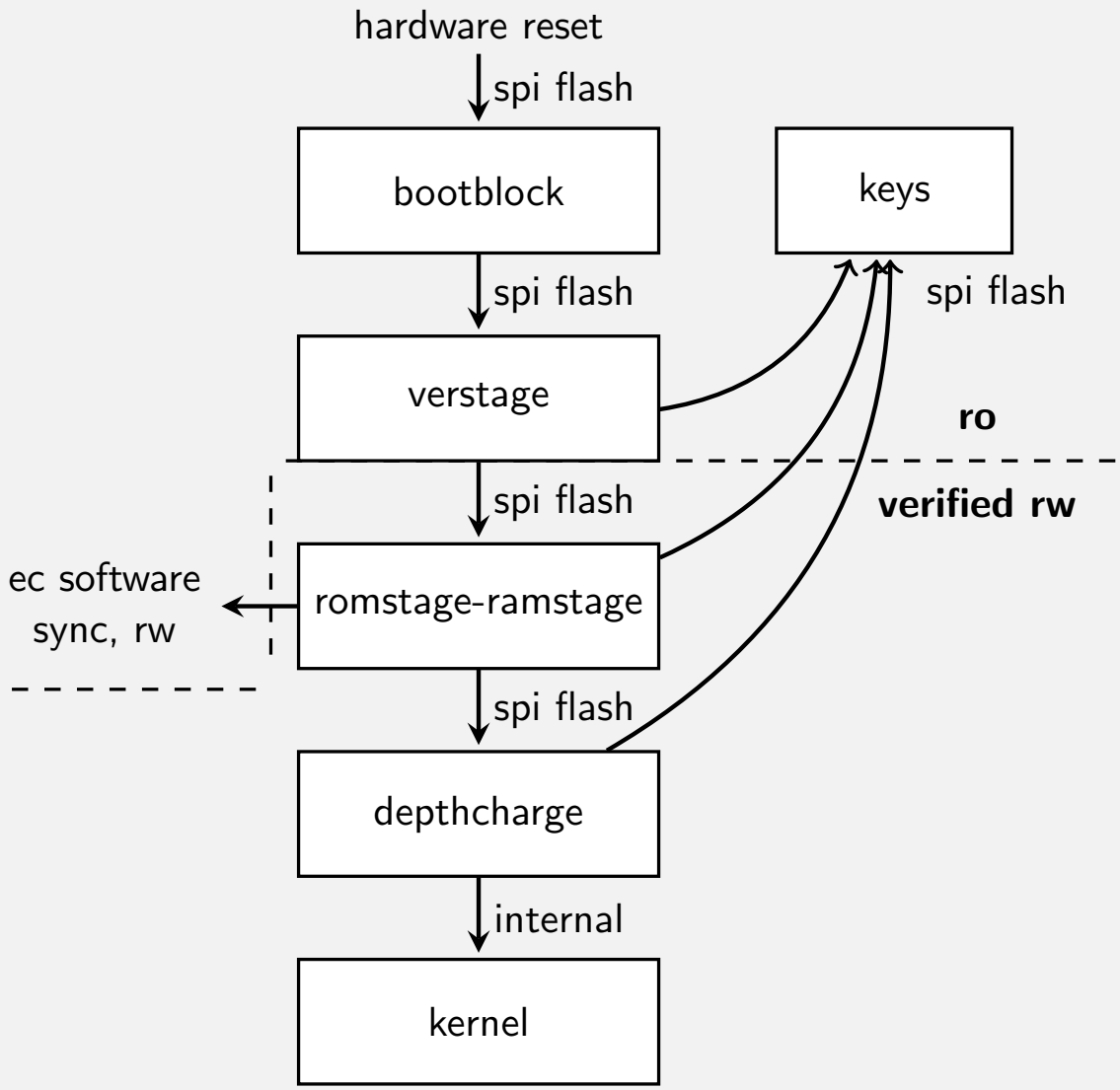
Replacing software and keys:



Normal Boot

Boot path:

- RO bootblock
- RO verstage
- RW next stages
- RO to RW EC
- Internal kernel
- No interaction



Recovery Boot

Trigger:

- Verification error
- User request

Boot path:

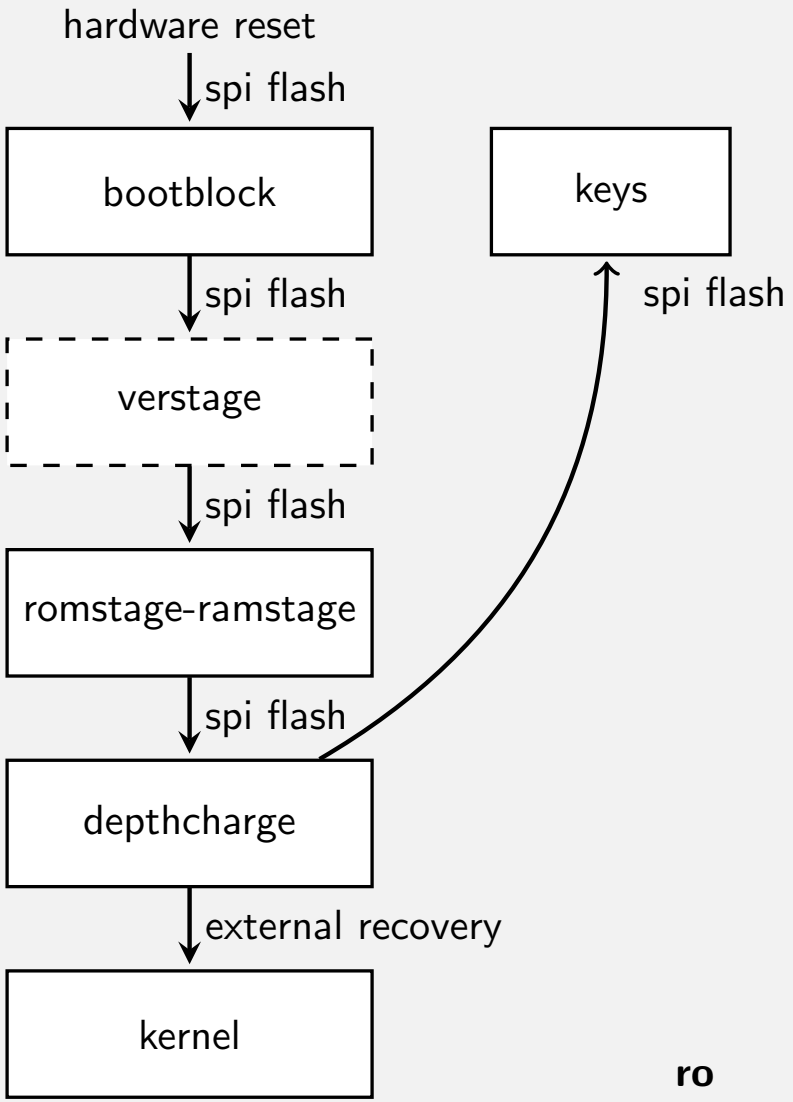
- RO only

Boot media:

- External recovery
- Recovery keys

Recovering:

- Instructions



Developer Boot

Enable:

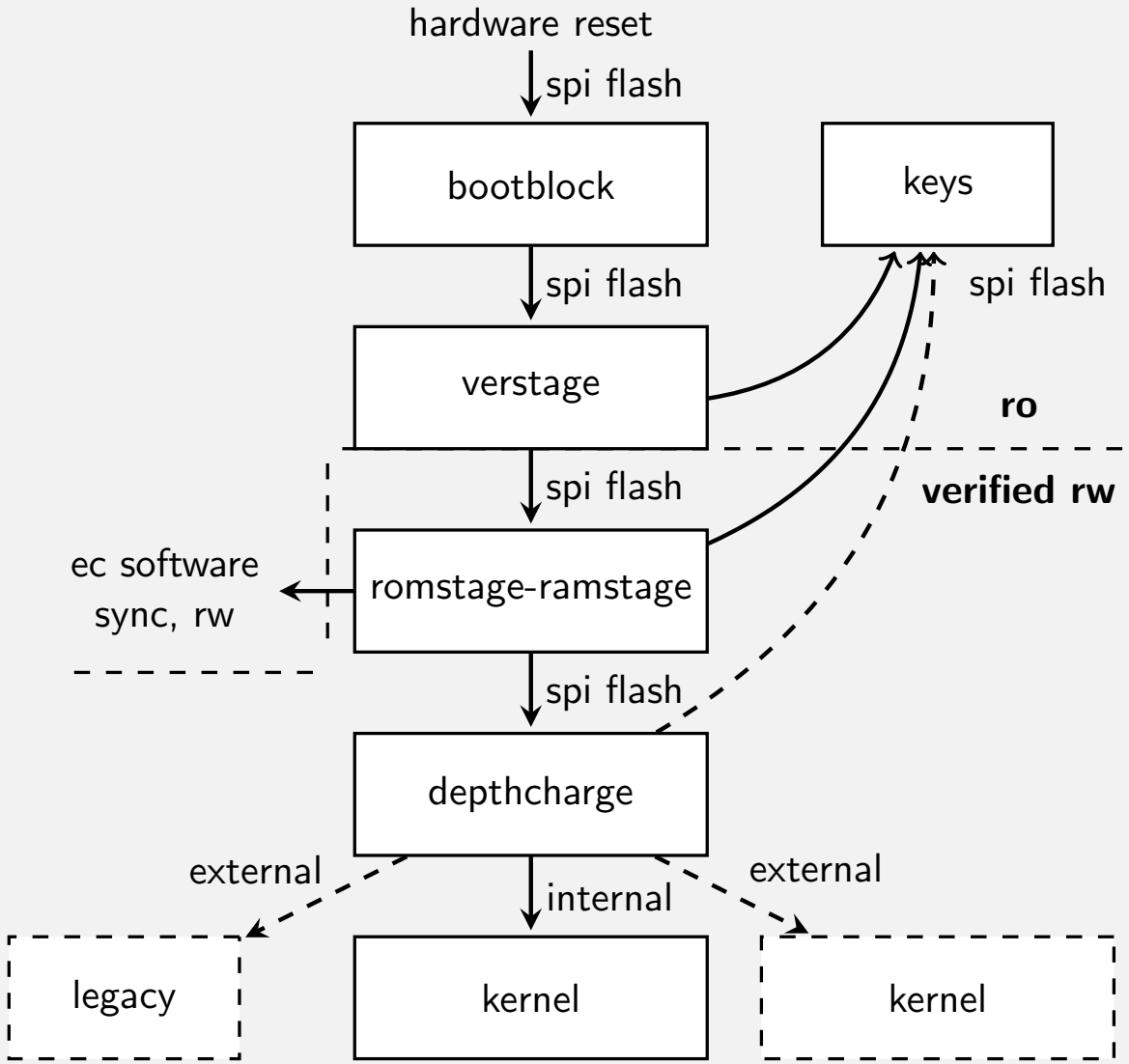
- Recovery mode
- Wipes data
- CrOS root
- Crossystem

Boot path:

- RO to RW
- Kernel verification

Boot media:

- Internal
- External
- Legacy



Devices

Hardware design constraints:

- SPI flash and the screw
- TPM
- Servo debug connector

Chromebooks, Chromeboxes, Chromebases, Chromebits

Platforms:

- **x86**: Intel Sandybridge, Haswell, Broadwell, Baytrail, Skylake
- **Signed ARM**: Samsung Exynos
- **Unsigned ARM**: Rockchip RK3288, nVidia Tegra K1

Unsigned ARM devices are great for freedom and privacy/security!

Community Support

CrOS developers community approach:

- *CrOS firmware team* history
- Friendly, helpful developers
- Contributions welcome, patch review
- Source code: `https://chromium.googlesource.com`

Community software:

- Upstream Coreboot support
- Libreboot support: build, images, documentation
- Upstream kernel support

Thank-You!

Questions?

Reference, interesting reads:

- https://www.chromium.org/chromium-os/chromiumos-design-docs
- https://blog.cr4.sh/2016/06/exploring-and-exploiting-lenovo.html
- https://coreboot.org/
- https://libreboot.org/
- https://firmwaresecurity.com/
- https://mjg59.dreamwidth.org/