EFI Secure Boot, shim and Xen

Current Status and Developments

Daniel Kiper
Software Developer, GRUB upstream maintainer

Platform Security Summit, May 23rd, 2018

Presentation Agenda

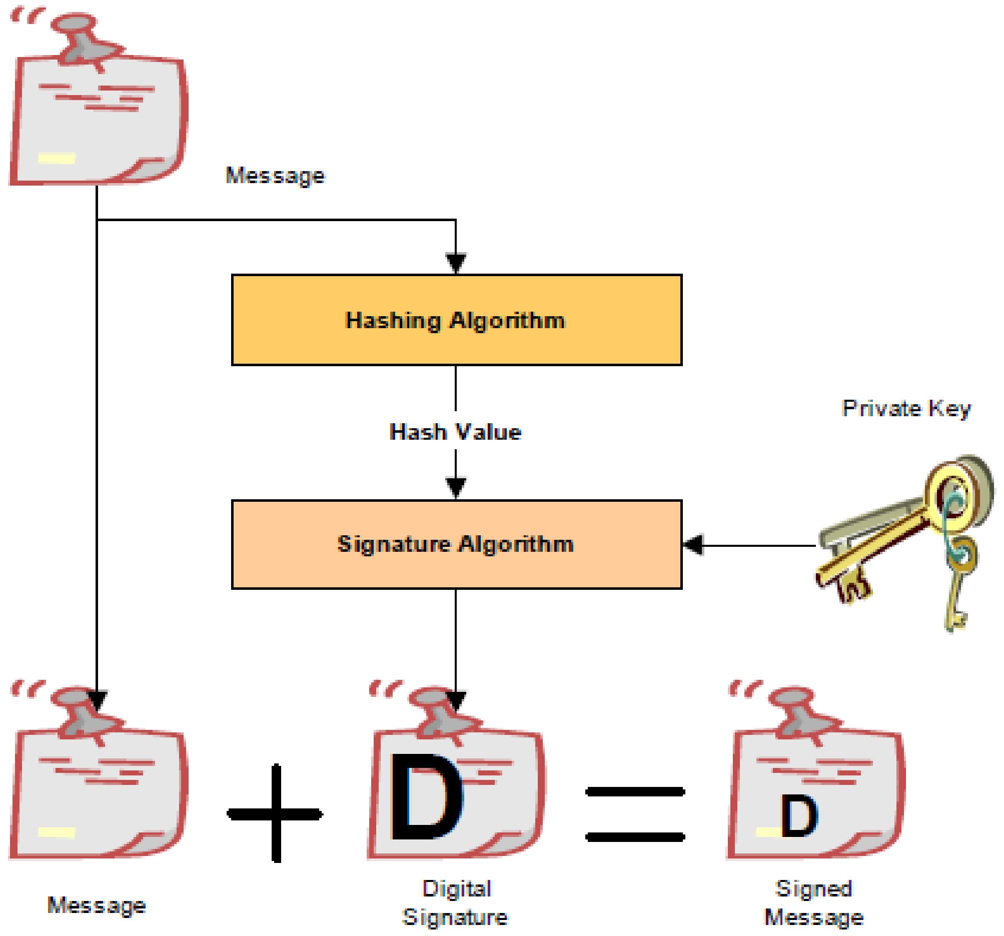
ORACLE

UEFI Secure Boot - General Overview

- UEFI secure boot was introduced in UEFI 2.0 spec,
- UEFI verifies the Portable Executable (PE) executables/drivers integrity before the execution and this way provides protection against some viruses, rootkits and other threats,
- Verification cannot be done outside the UEFI load/exec functions, i.e. it is not possible to verify the file/buffer using UEFI secure boot infrastructure on your own,
- UEFI spec does not impose any integrity requirements on the bootloaders, kernels, kernel modules, applications, etc., though some signing parties may impose some.

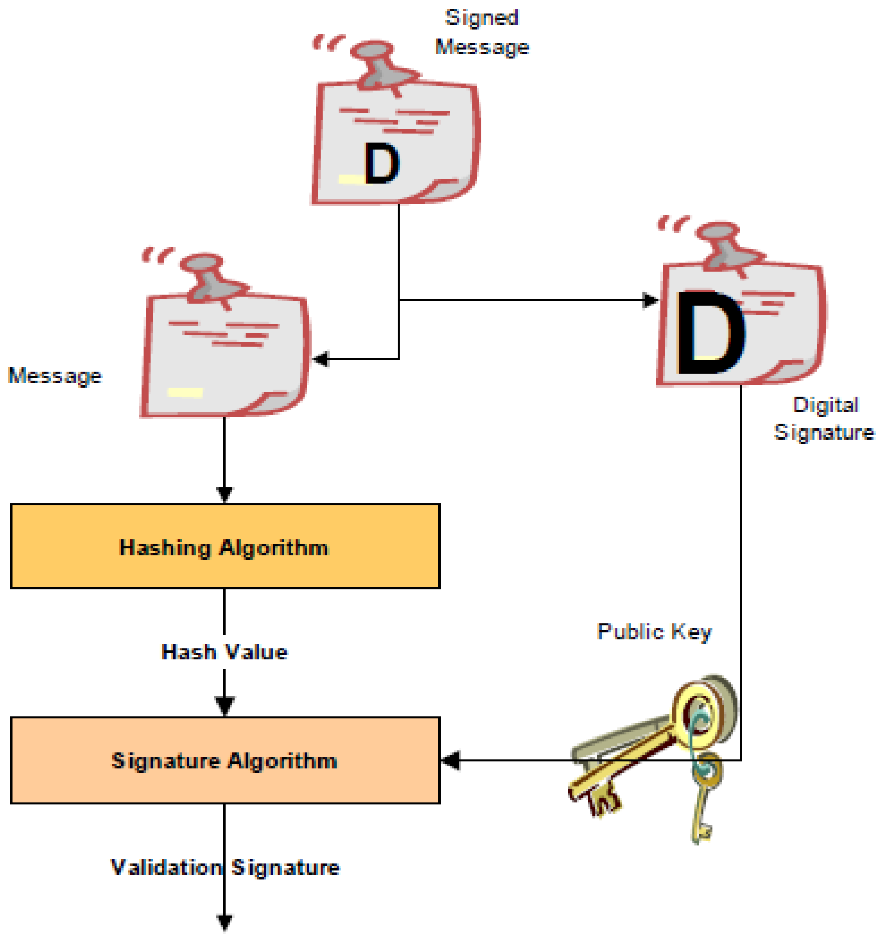
Creating a Digital Signature

Excerpt from UEFI 2.7 spec



Verifying a Digital Signature

Excerpt from UEFI 2.7 spec

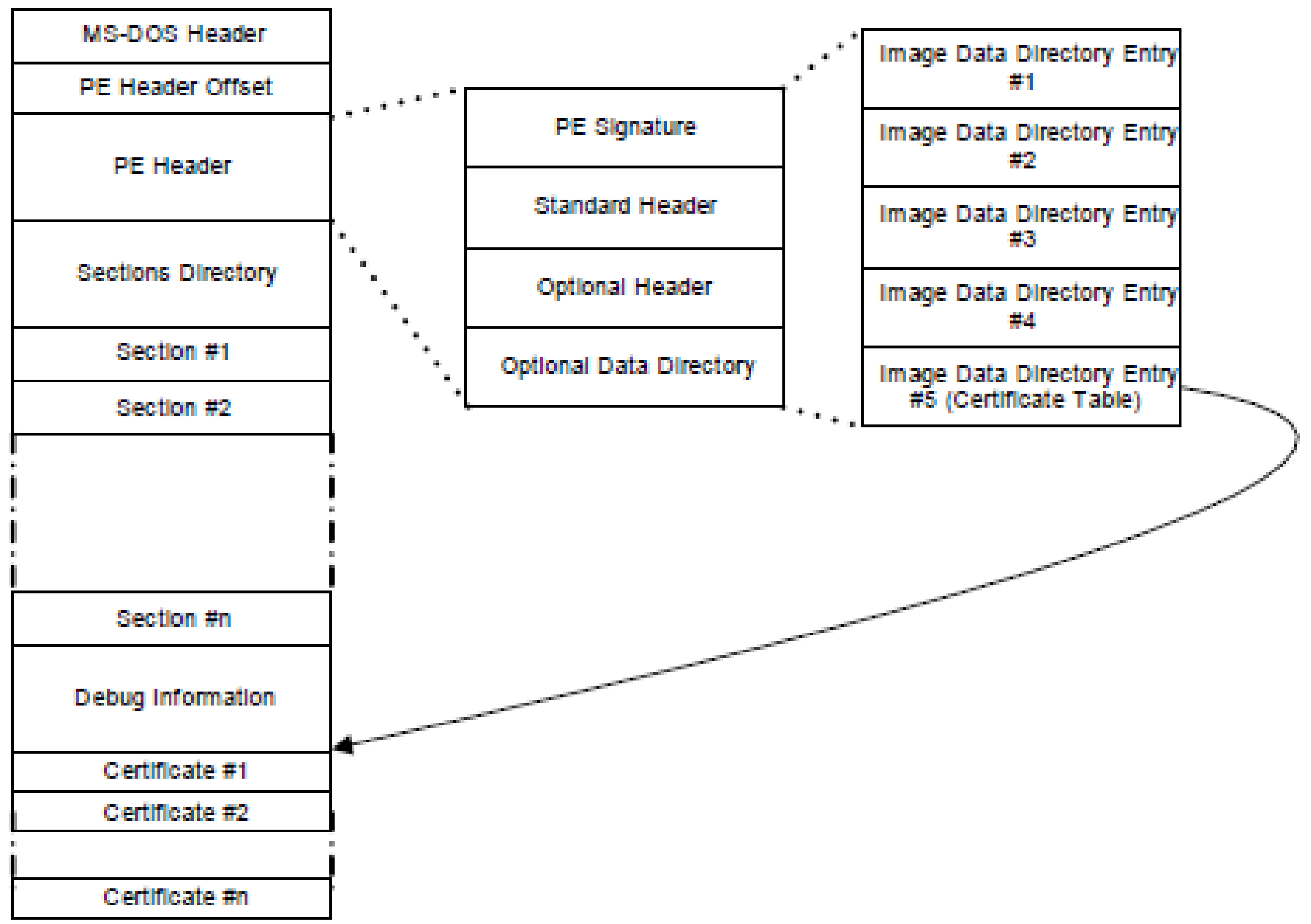


How generic signature algorithms apply to UEFI secure boot

- The message is a PE file loaded/executed by UEFI,
- The hashing algorithm belongs to a Secure Hash Algorithm (SHA) family,
- The signature algorithm is the X.509 (RFC 5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*) RSA-2048 (*PKCS #1 v2.2: RSA Cryptography Standard*),
- The signature can be attached to the PE file in PKCS #7 format (*PKCS #7: Cryptographic Message Syntax, Version 1.5*).

Embedding Digital Certificate/Signature into PE application

Excerpt from UEFI 2.7 spec



Digital Signature embedded into PE application

```
# objdump -x <signed_pe_file>


[…]
The Data Directory
Entry 0 0000000000000000 00000000 Export Directory [.edata (or where ever we found it)]
Entry 1 0000000000000000 00000000 Import Directory [parts of .idata]
Entry 2 0000000000000000 00000000 Resource Directory [.rsrc]
Entry 3 0000000000000000 00000000 Exception Directory [.pdata]
Entry 4 0000000000240040 000005d8 Security Directory
Entry 5 0000000000000000 00000000 Base Relocation Directory [.reloc]
[…]
```

PE Security Directory with PKCS #7 signature

```
typedef struct _WIN_CERTIFICATE {
  // The length of the entire certificate, including the length of the header, in bytes.
  UINT32 dwLength;
  UINT16 wRevision;         // 0x0200
  UINT16 wCertificateType;  // 0x0002 (WIN_CERT_TYPE_PKCS_SIGNED_DATA)
  //UINT8 bCertificate[ANYSIZE_ARRAY];
} WIN_CERTIFICATE;

# dd if=<signed_pe_file> of=signature.der bs=1 skip=2359368 count=1488
# openssl pkcs7 -inform DER -in signature.der -print_certs -text -noout
```

UEFI Secure Boot - Platform Key (PK)

Excerpt from UEFI 2.7 spec

The platform key establishes a trust relationship between the platform owner and the platform firmware. The platform owner enrolls the public half of the key (PKpub) into the platform firmware. The platform owner can later use the private half of the key (PKpriv) to change platform ownership or to enroll a Key Exchange Key. For UEFI, the recommended Platform Key format is RSA-2048.

…

The public key must be stored in non-volatile storage which is tamper and delete resistant.

UEFI Secure Boot - Key Exchange Key (KEK)

Excerpt from UEFI 2.7 spec

Key exchange keys establish a trust relationship between the operating system and the platform firmware. Each operating system (and potentially, each 3rd party application which need to communicate with platform firmware) enrolls a public key (KEKpub) into the platform firmware.

…

The public key must be stored in non-volatile storage which is tamper resistant.

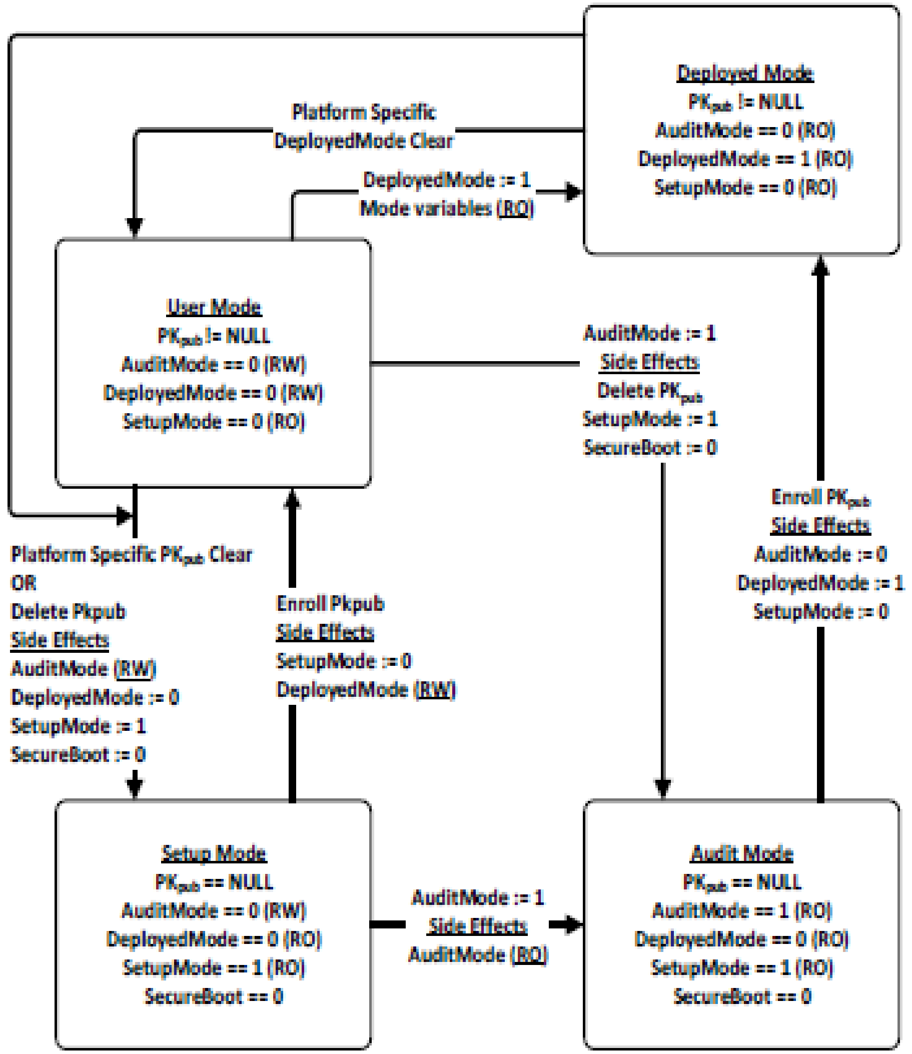
UEFI Secure Boot - db, dbx, dbt, dbr Databases

- db – the authorized signature database (hash or certificate),
- dbx – the forbidden signature database (hash or certificate),
- dbt – the timestamp signature database,
- dbr – the recovery signature database.

- *The signature database variables db, dbt, dbx, and dbr must be stored in tamper-resistant non-volatile storage.* (UEFI 2.7 spec)

UEFI Secure Boot Modes of Operation

Excerpt from UEFI 2.7 spec



UEFI Secure Boot Deficiencies

- Only signed drivers, bootloaders, kernels, etc. by well known signing parties (e.g. Microsoft) can be run on the most UEFI platforms; Some vendors, e.g. Oracle, provide their own keys for own platforms and OSes,
- Signing process is cumbersome and lengthy,
- The UEFI secure boot infrastructure is only available through the UEFI load/exec functions which usually are not used by the bootloaders or kernels to load additional modules, etc. Hence, it is not easy to use it outside the UEFI,
- GnuPG signatures are not supported.

shim – Signed Minimal Bootloader

- shim is the signed PE application whose sole purpose is to be a minimal bootloader which loads, verifies integrity and starts a full blown bootloader or an OS kernel,
- shim is able to verify the signatures and hashes looking into the UEFI secure boot databases, db*, into its own Machine Owner Key (MOK) database and into .vendor_cert PE section of itself,
- shim provides the UEFI protocol, known as SHIM_LOCK (SHIM_LOCK_GUID: {0x605dab50, 0xe046, 0x4300, {0xab, 0xb6, 0x3d, 0xd8, 0x10, 0xdd, 0x8b, 0x23}}, which can be used by the bootloaders, kernels, etc. to verify integrity of the loaded modules, etc.,
- The SHIM_LOCK protocol is not used by the UEFI load/exec functions,
- shim is provided by major Linux distributions.

Linux Foundation PreLoader – Signed Minimal Bootloader

- Linux Foundation PreLoader is another signed PE application whose sole purpose is to be a minimal bootloader which loads, verifies integrity and starts a full blown bootloader or an OS kernel,
- *The LF loader hooks into the low-level security architecture* (it uses UEFI Platform Initialization interface) *and installs its own handlers, which means the standard UEFI interfaces work. The upshot is that you can use bootloaders like Gummiboot or efilinux without having to modify them to call out to Shim.* Matthew Garrett (mjg59)
- Linux Foundation PreLoader is able to verify hashes only from Machine Owner Key (MOK) database,
- There were plans to merge the shim and the PreLoader functionality,
- Linux Foundation PreLoader is provided by some Linux distributions, e.g. Arch Linux.

shim – Xen and Linux dom0 kernel verification chain

- shim integrity is verified by the UEFI secure boot and if verification succeeds, then it is executed,
- shim checks the integrity of xen.efi using the SHIM_LOCK protocol and if check succeeds, then it is executed,
- xen.efi verifies the integrity of the dom0 kernel using the SHIM_LOCK protocol and if verification succeeds, then it is executed,
- dom0 kernel should check the modules integrity using its own methods, e.g. X.509 or GnuPG signatures.

shim – GRUB2, Xen and Linux dom0 kernel verification chain

- shim integrity is verified by the UEFI secure boot and if verification succeeds, then it is executed,
- shim checks the integrity of GRUB2 core using the SHIM_LOCK protocol and if check succeeds, then it is executed,
- *GRUB2 verifies the integrity of xen.efi using the SHIM_LOCK protocol and if verification succeeds, then it is executed via Multiboot2 protocol (this is under development),*
- xen.efi checks the integrity of dom0 kernel using the SHIM_LOCK protocol and if check succeeds, then it is executed,
- dom0 kernel should verify the modules integrity using its own methods, e.g. X.509 or GnuPG signatures.

shim – GRUB2 changes needed

- Add the verifiers framework (git://git.savannah.gnu.org/grub.git phcoder/verifiers branch),
- Build the SHIM_LOCK protocol verification functions on top of verifiers framework,
- Disable the GRUB2 modules load/unload,
- Disable the dangerous modules, e.g. iorw, memrw,
- https://lists.xen.org/archives/html/xen-devel/2017-07/msg00985.html
- *Note: The SHIM_LOCK protocol verification code (e.g. linuxefi command) provided in GRUB2 by various distros does not use verifiers framework.*

shim – Xen changes needed

- Add the Multiboot address and entry_addr fields,
- Add the Multiboot2 MULTIBOOT2_HEADER_TAG_ADDRESS and MULTIBOOT2_HEADER_TAG_ENTRY_ADDRESS tags,
- Add the the PE header to separate C file or xen/arch/x86/boot/head.S,
- Extract the SHIM_LOCK verification to separate function and call it from efi_multiboot2(),
- Change the relocation method for the initial page table entries,
- Generate the xen.efi from xen-syms using objcopy,
- https://lists.xen.org/archives/html/xen-devel/2017-07/msg00982.html

Xen boot and SHIM_LOCK changes - Gains

- One binary which can be loaded by the EFI loader, Multiboot and Multiboot2 protocols,
- If we wish, in the future we can drop xen/xen.gz and build xen.efi only,
- Crash dumps generated by the xen.efi loaded from the EFI loader can be analyzed by crash tool,
- Simpler code,
- Simpler build,
- Xen build no longer depends on ld i386pep support.

ORACLE

shim – UEFI x64 Platform Manual Setup - Example

- Install the following packages: efibootmgr, openssl, mokutil, shim, shim-signed, sbsigntool or pesign
- `# cp /usr/lib/shim/shim64.efi.signed /boot/efi/efi/shim/shim64.efi`
- `# cp /usr/lib/shim/mm64.efi.signed /boot/efi/efi/shim/mm64.efi`
- `# cp /usr/lib/shim/fb64.efi.signed /boot/efi/efi/shim/fb64.efi`
- `# efibootmgr -cL SHIM -d /dev/sda -p 2 -l '\efi\shim\shim.efi'`
- Build your UEFI bootloader, kernel, Xen or anything else and name it grubx64.efi

-
  ```
  # openssl req -new -x509 -newkey rsa:2048 -days 5000 \
      -subj '/CN=SHIM TEST/' -passout pass:<password> \
      -out shim-test-cert.pem -keyout shim-test-key.pem
  ```

-
  ```
  # openssl x509 -outform DER -in shim-test-cert.pem \
      -out shim-test-cert.crt
  ```

shim – UEFI x64 Platform Manual Setup – Example

- `# mokutil -import shim-test-cert.crt`
- `# sbsign --cert shim-test-cert.pem --key shim-test-key.pem grubx64.efi`
- `# cp grubx64.efi.signed /boot/efi/efi/shim/grubx64.efi`
- Reboot the machine and enroll „CN=SHIM TEST” public key into MOK database
- Enable the UEFI secure boot

Documentation & Resources

- Unified Extensible Firmware Interface Specification, Version 2.7
- http://www.uefi.org/
- http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx
- https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/windows-secure-boot-key-creation-and-management-guidance
- https://en.altlinux.org/UEFI_SecureBoot_mini-HOWTO
- http://www.rodsbooks.com/efi-bootloaders/secureboot.htm
- https://mjg59.dreamwidth.org/23113.html

Documentation & Resources

- https://www.cs.cornell.edu/courses/cs5430/2015sp/notes/rsa_sign_vs_dec.php
- https://www.itu.int/rec/T-REC-X.509
- https://tools.ietf.org/html/rfc5280
- https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-rsa-cryptography-standard.htm
- https://tools.ietf.org/html/rfc3447
- https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-7-cryptographic-message-syntax-standar.htm
- https://tools.ietf.org/html/rfc2315
- FIPS 180-4, Secure Hash Standard (SHS)

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Integrated Cloud

Applications & Platform Services