Configuring the bootloader

From Gentoo Wiki

< Handbook:AMD64 (/wiki/Special:MyLanguage/Handbook:AMD64) | Installation (/wiki/Special:MyLanguage/Handbook:AMD64/Installation)

Jump to:navigation Jump to:search

AMD64 Handbook (/wiki/Handbook:AMD64)
Installation (/wiki/Handbook:AMD64/Full/Installation)
About the installation (/wiki/Handbook:AMD64/Installation/About)
Choosing the media (/wiki/Handbook:AMD64/Installation/Media)
Configuring the network (/wiki/Handbook:AMD64/Installation/Networking)
Preparing the disks (/wiki/Handbook:AMD64/Installation/Disks)
The stage file (/wiki/Handbook:AMD64/Installation/Stage)
Installing base system (/wiki/Handbook:AMD64/Installation/Base)
Configuring the kernel (/wiki/Handbook:AMD64/Installation/Kernel)
Configuring the system (/wiki/Handbook:AMD64/Installation/System)
Installing tools (/wiki/Handbook:AMD64/Installation/Tools)
Configuring the bootloader
Finalizing (/wiki/Handbook:AMD64/Installation/Finalizing)
Working with Gentoo (/wiki/Handbook:AMD64/Full/Working)
Portage introduction (/wiki/Handbook:AMD64/Working/Portage)
USE flags (/wiki/Handbook:AMD64/Working/USE)
Portage features (/wiki/Handbook:AMD64/Working/Features)
Initscript system (/wiki/Handbook:AMD64/Working/Initscripts)
Environment variables (/wiki/Handbook:AMD64/Working/EnvVar)

Contents

Selecting a boot loader

With the Linux kernel configured, system tools installed and configuration files edited, it is time to install the last important piece of a Linux installation: the boot loader.

The boot loader is responsible for firing up the Linux kernel upon boot - without it, the system would not know how to proceed when the power button has been pressed.

For **`amd64`**, we document how to configure either GRUB or LILO for DOS/Legacy BIOS based systems, and GRUB, systemd-boot or EFI Stub for UEFI systems.

In this section of the Handbook a delineation has been made between *emerging* the boot loader's package and *installing* a boot loader to a system disk. Here the term *emerge* will be used to ask Portage (/wiki/Portage) to make the software package available to the system. The term *install* will signify the boot loader copying files or physically modifying appropriate sections of the system's disk drive in order to render the boot loader *activated and ready to operate* on the next power cycle.

Default: GRUB

By default, the majority of Gentoo systems now rely upon GRUB (/wiki/GRUB) (found in the `sys-boot/grub` (https://packages.gentoo.org/packages/sys-boot/grub) package), which is the direct successor to GRUB Legacy (/wiki/GRUB_Legacy). With no additional configuration, GRUB gladly supports older BIOS ("pc") systems. With a small amount of configuration, necessary before build time, GRUB can support more than a half a dozen additional platforms. For more information, consult the Prerequisites section (/wiki/GRUB#Prerequisites) of the GRUB (/wiki/GRUB) article.

Emerge

When using an older BIOS system supporting only MBR partition tables, no additional configuration is needed in order to emerge GRUB:

```
root # emerge --ask --verbose sys-boot/grub
```

A note for UEFI users: running the above command will output the enabled *GRUB_PLATFORMS* values before emerging. When using UEFI capable systems, users will need to ensure `GRUB_PLATFORMS="efi-64"` is enabled (as it is the case by default). If that is not the case for the setup, `GRUB_PLATFORMS="efi-64"` will need to be added to the **/etc/portage/make.conf** file *before* emerging GRUB so that the package will be built with EFI functionality:

```
root # echo 'GRUB_PLATFORMS="efi-64"' >> /etc/portage/make.conf
root # emerge --ask sys-boot/grub
```

If GRUB was somehow emerged without enabling `GRUB_PLATFORMS="efi-64"`, the line (as shown above) can be added to **make.conf** and then dependencies for the world package set (/wiki/World_set_(Portage)) can be re-calculated by passing the `--update --newuse` options to **emerge**:

```
root # emerge --ask --update --newuse --verbose sys-boot/grub
```

The GRUB software has now been merged onto the *system*, but it has not yet been installed as a secondary *bootloader*.

Install

Next, install the necessary GRUB files to the **/boot/grub/** directory via the **grub-install** command. Presuming the first disk (the one where the system boots from) is **/dev/sda**, one of the following commands will do:

DOS/Legacy BIOS systems

For DOS/Legacy BIOS systems:

```
root # grub-install /dev/sda
```

UEFI systems

Important
Make sure the EFI system partition has been mounted *before* running **grub-install**. It is possible for **grub-install** to install the GRUB EFI file (**grubx64.efi**) into the wrong directory **without** providing *any* indication the wrong directory was used.

For UEFI systems:

```
root # grub-install --efi-directory=/efi
```

```
Installing for x86_64-efi platform.
Installation finished. No error reported.
```

Upon successful installation, the output should match the output of the previous command. If the output does not match exactly, then proceed to Debugging GRUB (/wiki/Handbook:AMD64/Blocks/Bootloader#Debugging_GRUB), otherwise jump to the Configure step (/wiki/Handbook:AMD64/Blocks/Bootloader#GRUB_Configure).

Optional: Secure Boot

To successfully boot with secure boot enabled the signing certificate must either be accepted by the UEFI (/wiki/UEFI) firmware, or shim (/wiki/Shim) must be used as a pre-loader. Shim is pre-signed with the third-party Microsoft Certificate, accepted by default by most UEFI motherboards.

How to configure the UEFI firmware to accept custom keys depends on the firmware vendor, which is beyond the scope of the handbook. Below is shown how to setup shim instead. Here it is assumed that the user has already followed the instructions in the previous sections to generate a signing key and to configure portage to use it. If this is not the case please return first to the Kernel installation (/wiki/Handbook:AMD64/Installation/Kernel) section.

The package `sys-boot/grub` (https://packages.gentoo.org/packages/sys-boot/grub) installs a prebuilt and signed stand-alone EFI executable if the `secureboot (https://packages.gentoo.org/useflags/secureboot) (/wiki/USE_flag)` USE flag is enabled. Install the required packages and copy the stand-alone grub, Shim, and the MokManager to the same directory on the EFI System Partition. For example:

```
root # emerge sys-boot/grub sys-boot/shim sys-boot/mokutil sys-boot/efibootmgr
root # cp /usr/share/shim/BOOTX64.EFI /efi/EFI/Gentoo/shimx64.efi
root # cp /usr/share/shim/mmx64.efi /efi/EFI/Gentoo/mmx64.efi
root # cp /usr/lib/grub/grub-x86_64.efi.signed /efi/EFI/Gentoo/grubx64.efi
```

Next register the signing key in shims MOKlist, this requires keys in the *DER* format, whereas **sbsign** and the kernel build system expect keys in the *PEM* format. In the previous sections of the handbook an example was shown to generate such a signing *PEM* key, this key must now be converted to the *DER* format:

```
root # openssl x509 -in /path/to/kernel_key.pem -inform PEM -out /path/to/kernel_key.der -outform DER
```

Note
The path used here must be the path to the pem file containing the certificate belonging to the generated key. In this example both key and certificate are in the same pem file.

Then the converted certificate can be imported into Shims MOKlist, this command will ask to set some password for the import request:

```
root # mokutil --import /path/to/kernel_key.der
```

Note
When the currently booted kernel already trusts the certificate being imported, the message "Already in kernel trusted keyring." will be returned here. If this happens, re-run the above command with the argument **`--ignore-keyring`** added.

Next, register Shim with the UEFI firmware. In the following command, `boot-disk` and `boot-partition-id` must be replaced with the disk and partition identifier of the EFI system partition:

```
root # efibootmgr --create --disk /dev/boot-disk --part boot-partition-id --loader '\EFI\Gentoo
\shimx64.efi' --label 'GRUB via Shim' --unicode
```

Note that this prebuilt and signed stand-alone version of grub reads the *grub.cfg* from a different location then usual. Instead of the default **`/boot/grub/grub.cfg`** the config file should be in the same directory that the grub EFI executable is in, e.g. **`/efi/EFI/Gentoo/grub.cfg`**. When `sys-kernel/installkernel` (https://packages.gentoo.org/packages/sys-kernel/installkernel) is used to install the kernel and update the grub configuration then the *GRUB_CFG* environment variable may be used to override the usual location of the grub config file.

For example:

```
root # grub-mkconfig -o /efi/EFI/Gentoo/grub.cfg
```

Or, via installkernel (/wiki/Installkernel):

FILE **`/etc/env.d/99grub`**

```
GRUB_CFG=/efi/EFI/Gentoo/grub.cfg
```

```
root # env-update
```

Note
The import process will not be completed until the system is rebooted. After completing all steps in the handbook, restart the system and Shim will load, it will find the import request registered by **`mokutil`**. The MokManager application will start and ask for the password that was set when creating the import request. Follow the on-screen instructions to complete the import of the certificate, then reboot the system into the UEFI menu and enable the Secure Boot setting.

Debugging GRUB

When debugging GRUB, there are a couple of quick fixes that may result in a bootable installation without having to reboot to a new live image environment.

In the event that "EFI variables are not supported on this system" is displayed somewhere in the output, it is likely the live image was not booted in EFI mode and is presently in Legacy BIOS boot mode. The solution is to try the removable GRUB step (/wiki/Handbook:AMD64/Blocks/Bootloader#GRUB_Install_EFI_systems_removable) mentioned below. This will overwrite the executable EFI file located at **`/EFI/BOOT/BOOTX64.EFI`**. Upon rebooting in EFI mode, the motherboard firmware may execute this default boot entry and execute GRUB.

If **`grub-install`** returns an error that says "Could not prepare Boot variable: Read-only file system", and the live environment was correctly booted in UEFI mode, then it should be possible to remount the efivars special mount as read-write and then re-run the aforementioned **`grub-install`** command (/wiki/Handbook:AMD64/Blocks/Bootloader#GRUB_Install_EFI_systems_command):

```
root # mount -o remount,rw,nosuid,nodev,noexec --types efivarfs efivarfs /sys/firmware/efi/efivars
```

This is caused by certain non-official Gentoo environments not mounting the special EFI filesystem by default. If the previous command does not run, then reboot using an official Gentoo live image environment in EFI mode.

Some motherboard manufacturers with poor UEFI implementations seem to *only* support the **`/EFI/BOOT`** directory location for the .EFI file in the EFI System Partition (ESP). The GRUB installer can create the .EFI file in this location automatically by appending the `--removable` option to the install command. Ensure the ESP has been mounted before running the following command; presuming it is mounted at **`/efi`** (as defined earlier), run:

```
root # grub-install --target=x86_64-efi --efi-directory=/efi --removable
```

This creates the 'default' directory defined by the UEFI specification, and then creates a file with the default name: **`BOOTX64.EFI`**.

Configure

Next, generate the GRUB configuration based on the user configuration specified in the **/etc/default/grub** file and **/etc/grub.d** scripts. In most cases, no configuration is needed by users as GRUB will automatically detect which kernel to boot (the highest one available in **/boot/**) and what the root file system is. It is also possible to append kernel parameters in **/etc/default/grub** using the *GRUB_CMDLINE_LINUX* variable.

To generate the final GRUB configuration, run the **grub-mkconfig** command:

```
root # grub-mkconfig -o /boot/grub/grub.cfg
```

```
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-6.6.21-gentoo
Found initrd image: /boot/initramfs-genkernel-amd64-6.6.21-gentoo
done
```

The output of the command must mention that at least one Linux image is found, as those are needed to boot the system. If an initramfs is used or **genkernel** was used to build the kernel, the correct initrd image should be detected as well. If this is not the case, go to **/boot/** and check the contents using the **ls** command. If the files are indeed missing, go back to the kernel configuration and installation instructions.

Tip
The **os-prober** utility can be used in conjunction with GRUB to detect other operating systems from attached drives. Windows 7, 8.1, 10, and other distributions of Linux are detectable. Those desiring dual boot systems should emerge the `sys-boot/os-prober` (https://packages.gentoo.org/packages/sys-boot/os-prober) package then re-run the **grub-mkconfig** command (as seen above). If detection problems are encountered be sure to read the GRUB (/wiki/GRUB) article in its entirety *before* asking the Gentoo community for support.

Alternative 1: LILO

Emerge

LILO, the **LI**nux**LO**ader, is the tried and true workhorse of Linux boot loaders. However, it lacks features when compared to GRUB. LILO is still used because, on some systems, GRUB does not work and LILO does. Of course, it is also used because some people know LILO and want to stick with it. Either way, Gentoo supports both bootloaders.

Installing LILO is a breeze; just use emerge.

```
root # emerge --ask sys-boot/lilo
```

Configure

To configure LILO, first create **/etc/lilo.conf**:

```
root # nano -w /etc/lilo.conf
```

In the configuration file, sections are used to refer to the bootable kernel. Make sure that the kernel files (with kernel version) and initramfs files are known, as they need to be referred to in this configuration file.

Note
If the root filesystem is JFS, add an `append="ro"` line after each boot item since JFS needs to replay its log before it allows read-write mounting.

FILE **/etc/lilo.conf** **Example LILO configuration**

```
boot=/dev/sda             # Install LILO in the MBR
prompt                    # Give the user the chance to select another section
timeout=50                # Wait 5 (five) seconds before booting the default section
default=gentoo            # When the timeout has passed, boot the "gentoo" section
compact                   # This drastically reduces load time and keeps the map file smaller; may f
ail on some systems

image=/boot/vmlinuz-6.6.21-gentoo
  label=gentoo            # Name we give to this section
  read-only               # Start with a read-only root. Do not alter!
  root=/dev/sda3          # Location of the root filesystem

image=/boot/vmlinuz-6.6.21-gentoo
  label=gentoo.rescue     # Name we give to this section
  read-only               # Start with a read-only root. Do not alter!
  root=/dev/sda3         # Location of the root filesystem
  append="init=/bin/bb"   # Launch the Gentoo static rescue shell

# The next two lines are for dual booting with a Windows system.
# In this example, Windows is hosted on /dev/sda6.
other=/dev/sda6
  label=windows
```

Note
If a different partitioning scheme and/or kernel image is used, adjust accordingly.

If an initramfs is necessary, then change the configuration by referring to this initramfs file and telling the initramfs where the root device is located:

FILE **`/etc/lilo.conf`** **Adding initramfs information to a boot entry**

```
image=/boot/vmlinuz-6.6.21-gentoo
  label=gentoo
  read-only
  append="root=/dev/sda3"
  initrd=/boot/initramfs-genkernel-amd64-6.6.21-gentoo
```

If additional options need to be passed to the kernel, use an `append` statement. For instance, to add the `video` statement to enable framebuffer:

FILE **`/etc/lilo.conf`** **Adding video parameter to the boot options**

```
image=/boot/vmlinuz-6.6.21-gentoo
  label=gentoo
  read-only
  root=/dev/sda3
  append="video=uvesafb:mtrr,ywrap,1024x768-32@85"
```

Users that used **`genkernel`** should know that their kernels use the same boot options as is used for the installation CD. For instance, if SCSI device support needs to be enabled, add `doscsi` as kernel option.

Now save the file and exit.

Install

To finish up, run the **`/sbin/lilo`** executable so LILO can apply the **`/etc/lilo.conf`** settings to the system (i.e. install itself on the disk). Keep in mind that **`/sbin/lilo`** must be executed each time a new kernel is installed or a change has been made to the **`lilo.conf`** file in order for the system to boot if the filename of the kernel has changed.

```
root # /sbin/lilo
```

Alternative 2: EFI Stub

Computer systems with UEFI-based firmware technically do not need secondary bootloaders (e.g. GRUB) in order to boot kernels. Secondary bootloaders exist to *extend* the functionality of UEFI firmware during the boot process. Using GRUB (see the prior section) is typically easier and more robust because it offers a more flexible approach for quickly modifying kernel parameters at boot time.

System administrators who desire to take a minimalist, although more rigid, approach to booting the system can avoid secondary bootloaders and boot the Linux kernel as an EFI stub (/wiki/EFI_stub).

The `sys-boot/efibootmgr` (https://packages.gentoo.org/packages/sys-boot/efibootmgr) application is a tool to used interact with UEFI firmware - the system's primary bootloader. Normally this looks like adding or removing boot entries to the firmware's list of bootable entries. It can also update firmware settings so that the Linux kernels that were previously added as bootable entries can be executed with additional options. These interactions are performed through special data structures called EFI variables (hence the need for kernel support of EFI vars).

Ensure the EFI stub (/wiki/EFI_stub) kernel article has been reviewed *before* continuing. The kernel must have specific options enabled to be directly bootable by the UEFI firmware. It may be necessary to recompile the kernel in order to build-in this support.

It is also a good idea to take a look at the **`efibootmgr (/wiki/Efibootmgr)`** article for additional information.

Note
To reiterate, **`efibootmgr`** is *not* a requirement to boot an UEFI system; it is merely necessary to add an entry for an EFI-stub kernel into the UEFI firmware. When built appropriately with EFI stub support, the Linux kernel itself can be booted *directly*. Additional kernel command-line options can be built-in to the Linux kernel (there is a kernel configuration option called *CONFIG_CMDLINE*. Similarly, support for initramfs can be 'built-in' to the kernel as well.

To boot the kernel directly from the firmware, the kernel image must be present on the EFI System Partition (/wiki/EFI_System_Partition). This may be accomplished by enabling the `efistub (https://packages.gentoo.org/useflags/efistub)` `(/wiki/USE_flag)` USE flag on `sys-kernel/installkernel` (https://packages.gentoo.org/packages/sys-kernel/installkernel). Given that EFI Stub booting is not guaranteed to work on every UEFI system, the USE flag is stable masked, testing keywords must be accepted for installkernel to use this feature.

FILE **`/etc/portage/package.accept_keywords/installkernel`**

```
sys-kernel/installkernel **
sys-boot/uefi-mkconfig **
app-emulation/virt-firmware **
```

FILE **`/etc/portage/package.use/installkernel`**

```
sys-kernel/installkernel efistub
```

Then reinstall installkernel (/wiki/Installkernel), create the **`/efi/EFI/Gentoo`** directory and reinstall the kernel:

```
root # emerge --ask sys-kernel/installkernel
root # mkdir -p /efi/EFI/Gentoo
```

For distribution kernels:

```
root # emerge --ask --config sys-kernel/gentoo-kernel{,-bin}
```

For manually managed kernels:

```
root # make install
```

Alternatively, when installkernel (/wiki/Installkernel) is not used, the kernel may be copied manually to the EFI System Partition, calling it **`bzImage.efi`**:

```
root # mkdir -p /efi/EFI/Gentoo
root # cp /boot/vmlinuz-* /efi/EFI/Gentoo/bzImage.efi
```

Install the **`efibootmgr`** package:

```
root # emerge --ask sys-boot/efibootmgr
```

Create boot entry called "Gentoo" for the freshly compiled EFI stub kernel within the UEFI firmware:

```
root # efibootmgr --create --disk /dev/sda --part 1 --label "gentoo" --loader "\EFI\Gentoo\bzImage.efi"
```

Note
The use of a backslash (`\`) as directory path separator is mandatory when using UEFI definitions.

If an initial RAM file system (initramfs) is used, copy it to the EFI System Partition as well, then add the proper boot option to it:

```
root # efibootmgr --create --disk /dev/sda --part 1 --label "gentoo" --loader "\EFI\Gentoo\bzImage.efi"
--unicode "initrd=\EFI\Gentoo\initramfs.img"
```

Tip
Additional kernel command line options may be parsed by the firmware to the kernel by specifying them along with the *initrd=...* option as shown above.

With these changes done, when the system reboots, a boot entry called "gentoo" will be available.

Unified Kernel Image

If installkernel (/wiki/Installkernel) was configured to build and install unified kernel images. The unified kernel image should already be installed to the **EFI/Linux** directory on the EFI system partition, if this is not the case ensure the directory exists and then run the kernel installation again as described earlier in the handbook.

To add a direct boot entry for the installed unified kernel image:

```
root # efibootmgr --create --disk /dev/sda --part 1 --label "gentoo" --loader "\EFI\Linux\gentoo-x.y.z.efi"
```

Alternative 3: Syslinux

Syslinux is yet another bootloader alternative for the **amd64** architecture. It supports MBR and, as of version 6.00, it supports EFI boot. PXE (network) boot and lesser-known options are also supported. Although Syslinux is a popular bootloader for many it is unsupported by the Handbook. Readers can find information on emerging and then installing this bootloader in the Syslinux (/wiki /Syslinux) article.

Alternative 4: systemd-boot

Another option is systemd-boot (/wiki/Systemd/systemd-boot), which works on both OpenRC and systemd machines. It is a thin chainloader and works well with secure boot.

To install systemd-boot, enable the `boot (https://packages.gentoo.org/useflags/boot)` `(/wiki/USE_flag)` USE flag and re-install `sys-apps/systemd` (https://packages.gentoo.org/packages/sys-apps/systemd) (for systemd systems) or `sys-apps/systemd-utils` (https://packages.gentoo.org/packages/sys-apps/systemd-utils) (for OpenRC systems):

FILE **/etc/portage/package.use/systemd-boot**

```
sys-apps/systemd boot
sys-apps/systemd-utils boot
```

```
root # emerge --ask sys-apps/systemd
```

Or

```
root # emerge --ask sys-apps/systemd-utils
```

Then, install the systemd-boot loader to the EFI System Partition (/wiki/EFI_System_Partition):

```
root # bootctl install
```

Important
Make sure the EFI system partition has been mounted *before* running **bootctl install**.

When using this bootloader, before rebooting, verify that a new bootable entry exists using:

```
root # bootctl list
```

Warning
The kernel command line for new systemd-boot entries is read from **/etc/kernel/cmdline** or **/usr/lib/kernel /cmdline**. If neither file is present, then the kernel command line of the currently booted kernel is re-used (**/proc /cmdline**). On new installs it might therefore happen that the kernel command line of the live CD is accidentally used to boot the new kernel. The kernel command line for registered entries can be checked with:

```
root # bootctl list
```

If this does not show the desired kernel command line then create **/etc/kernel/cmdline** containing the correct kernel command line and re-install the kernel.

If no new entry exists, ensure the `sys-kernel/installkernel` (https://packages.gentoo.org/packages/sys-kernel/installkernel) package has been installed with the `systemd (https://packages.gentoo.org/useflags/systemd)` `(/wiki/USE_flag)` and `systemd-boot (https://packages.gentoo.org/useflags/systemd-boot)` `(/wiki/USE_flag)` USE flags enabled, and re-run the kernel installation.

For the distribution kernels:

```
root # emerge --ask --config sys-kernel/gentoo-kernel
```

For a manually configured and compiled kernel:

```
root # make install
```

Important
When installing kernels for systemd-boot, no **root=** kernel command line argument is added by default. On systemd systems that are using an initramfs users may rely instead on systemd-gpt-auto-generator (/wiki

/Systemd#Automatic_mounting_of_partitions_at_boot) to automatically find the root partition at boot. Otherwise users should manually specify the location of the root partition by setting **root=** in **/etc/kernel/cmdline** as well as any other kernel command line arguments that should be used. And then reinstalling the kernel as described above.

Optional: Secure Boot

When the `secureboot (https://packages.gentoo.org/useflags/secureboot)` `(/wiki/USE_flag)` USE flag is enabled, the systemd-boot EFI executable will be signed by portage automatically. Furthermore, **bootctl install** will automatically install the signed version.

To successfully boot with secure boot enabled the used certificate must either be accepted by the UEFI (/wiki/UEFI) firmware, or shim (/wiki/Shim) must be used as a pre-loader. Shim is pre-signed with the third-party Microsoft Certificate, accepted by default by most UEFI motherboards.

How to configure the UEFI firmware to accept custom keys depends on the firmware vendor, which is beyond the scope of the handbook. Below is shown how to setup shim instead. Here it is assumed that the user has already followed the instructions in the previous sections to generate a signing key and to configure portage to use it. If this is not the case please return first to the Kernel installation (/wiki/Handbook:AMD64/Installation/Kernel) section.

```
root # emerge --ask sys-boot/shim sys-boot/mokutil sys-boot/efibootmgr
root # bootctl install --no-variables
root # cp /usr/share/shim/BOOTX64.EFI /efi/EFI/systemd/shimx64.efi
root # cp /usr/share/shim/mmx64.efi /efi/EFI/systemd/mmx64.efi
```

Shims MOKlist requires keys in the *DER* format, whereas **sbsign** and the kernel build system expect keys in the *PEM* format. In the previous sections of the handbook an example was shown to generate such a signing *PEM* key, this key must now be converted to the *DER* format:

```
root # openssl x509 -in /path/to/kernel_key.pem -inform PEM -out /path/to/kernel_key.der -outform DER
```

Note
The path used here must be the path to the pem file containing the certificate belonging to the generated key. In this example both key and certificate are in the same pem file.

Then the converted certificate can be imported into Shims MOKlist:

```
root # mokutil --import /path/to/kernel_key.der
```

Note
When the currently booted kernel already trusts the certificate being imported, the message "Already in kernel trusted keyring." will be returned here. If this happens, re-run the above command with the argument **--ignore-keyring** added.

And finally we register Shim with the UEFI firmware. In the following command, `boot-disk` and `boot-partition-id` must be replaced with the disk and partition identifier of the EFI system partition:

```
root # efibootmgr --create --disk /dev/boot-disk --part boot-partition-id --loader '\EFI\systemd
\shimx64.efi' --label 'Systemd-boot via Shim' --unicode '\EFI\systemd\systemd-bootx64.efi'
```

Note
The import process will not be completed until the system is rebooted. After completing all steps in the handbook, restart the system and Shim will load, it will find the import request registered by **mokutil**. The MokManager application will start and ask for the password that was set when creating the import request. Follow the on-screen instructions to complete the import of the certificate, then reboot the system into the UEFI menu and enable the Secure Boot setting.

Rebooting the system

Exit the chrooted environment and unmount all mounted partitions. Then type in that one magical command that initiates the final, true test: **reboot**.

```
(chroot) livecd # exit
livecd~# cd
livecd~# umount -l /mnt/gentoo/dev{/shm,/pts,}
livecd~# umount -R /mnt/gentoo
livecd~# reboot
```

Do not forget to remove the live image, otherwise it may be targeted again instead of the newly installed Gentoo system!

Once rebooted in the fresh Gentoo environment, it is wise to finish with Finalizing the Gentoo installation (/wiki/Handbook:AMD64/Installation/Finalizing).

← Installing tools (/wiki/Handbook:AMD64/Installation/Tools) | Home (/wiki/Handbook:AMD64) | Finalizing → (/wiki/Handbook:AMD64/Installat

Retrieved from "https://wiki.gentoo.org/index.php?title=Handbook:AMD64/Installation/Bootloader&oldid=212430 (https://wiki.gentoo.org/index.php?title=Handbook:AMD64/Installation/Bootloader&oldid=212430)"