embedded projects GmbH

HARDWARE FOR PROJECTS

User Manual
Picosafe

August 24, 2013

Contents

Chapter 1

Quick Start

- Connect Picosafe with an usb cable to your pc
- Two mass storages appears (one with an start.html and a public open flash disk)
- Click on start.html
- Click on start link (to url https://172.24.42.1)
- Acknowledge the self signes SSL certificate
- With windows you have to first install the picosafe.inf (Please accept not trusted driver)
- Open the decrypted filesystem with the default password: picosafe
- After the stick is ready the main screen appears automatically (sometimes this needs some time)
- Connect you with ssh to change the default passwords

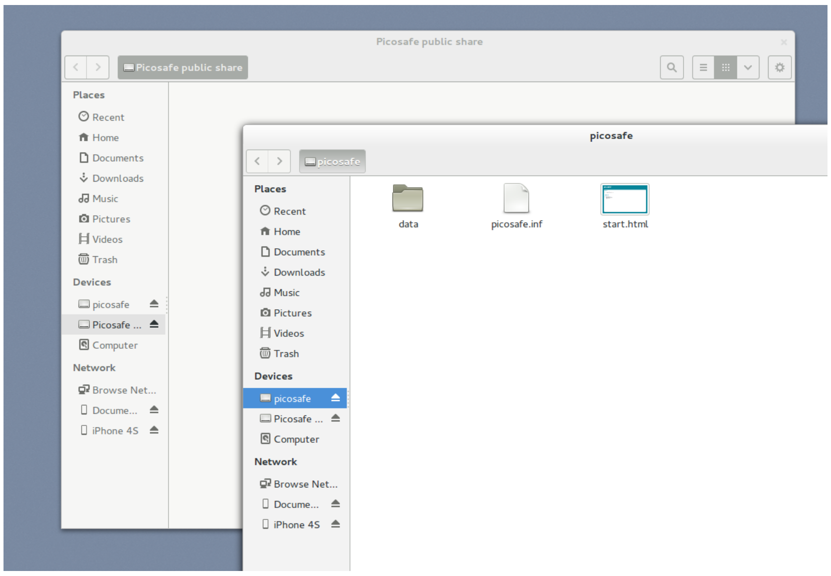
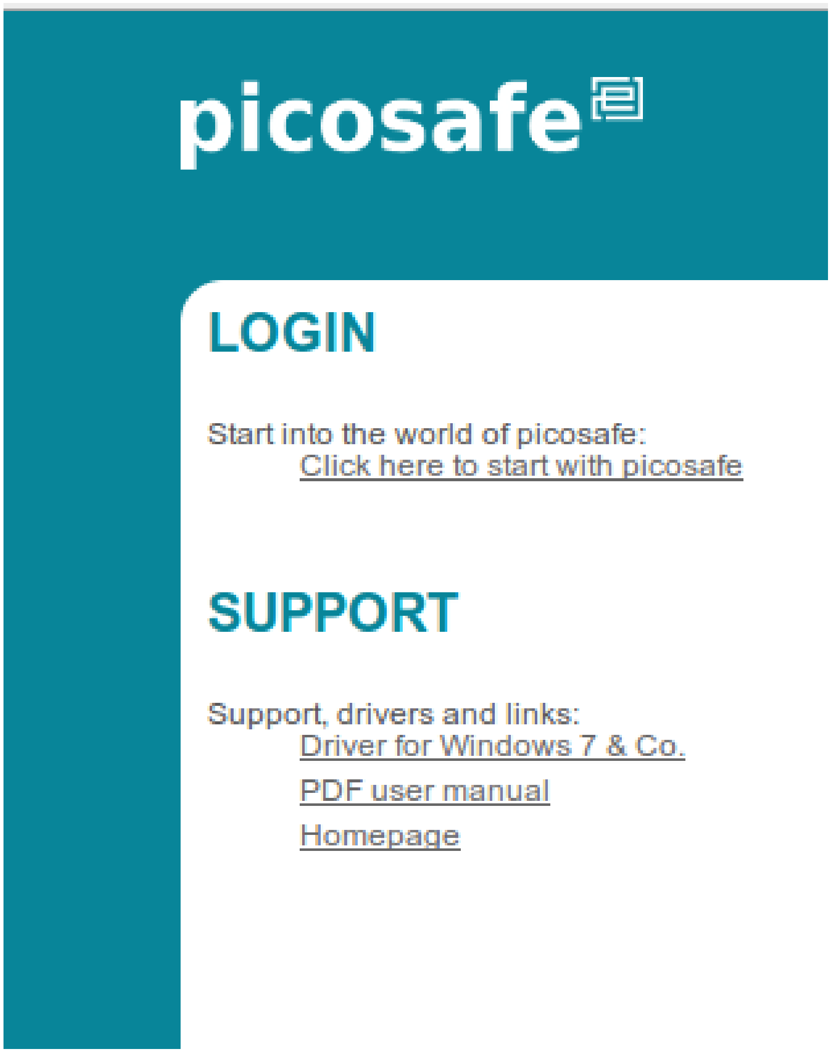


Figure 1.1: Drives after connect Picosafe to USB.


picosafe

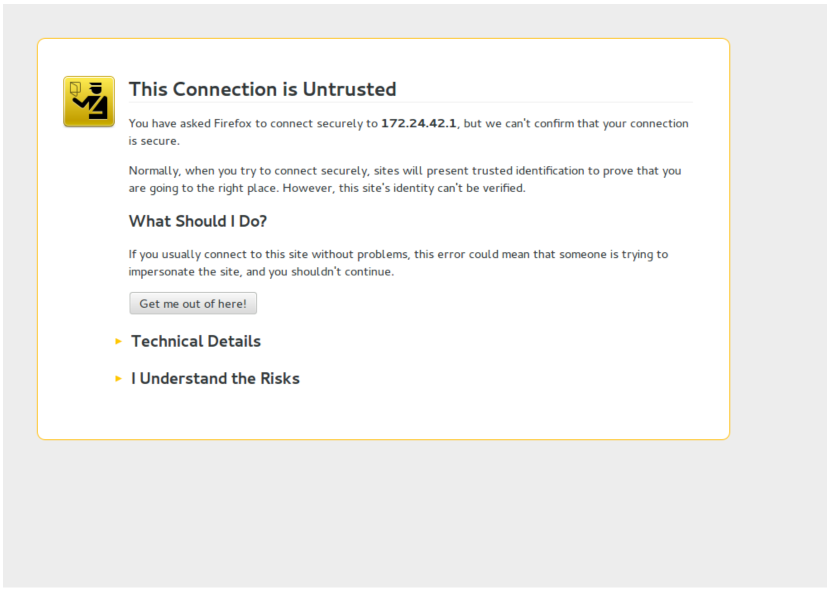
LOGIN

Start into the world of picosafe:
Click here to start with picosafe

SUPPORT

Support, drivers and links:
Driver for Windows 7 & Co.
PDF user manual
Homepage

Figure 1.2: Open start.html and click on start



This Connection is Untrusted

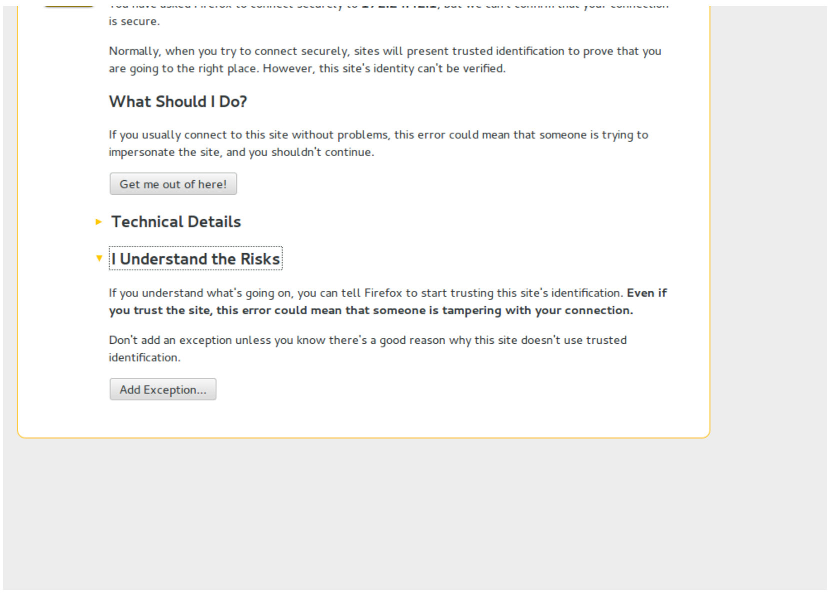
You have asked Firefox to connect securely to **172.24.42.1**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Figure 1.3: Click: I Understand the risk



is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

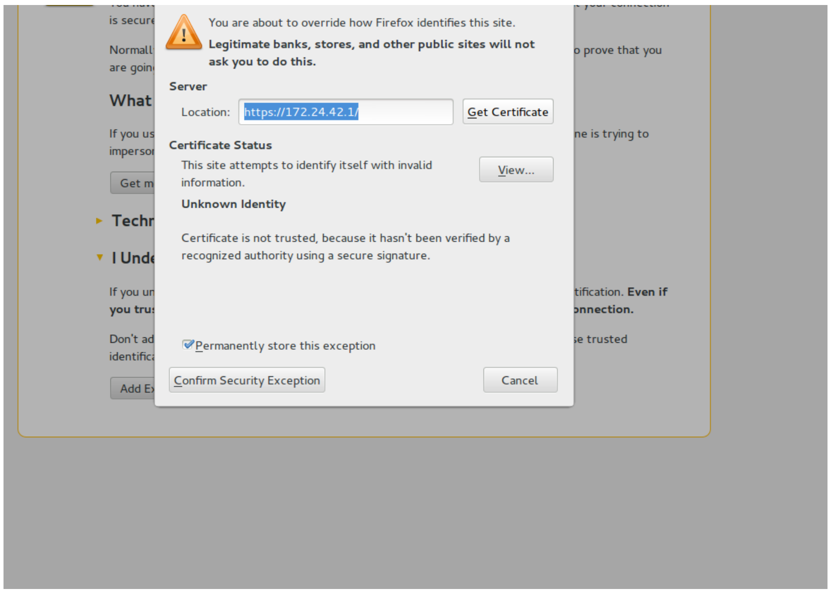
What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

Figure 1.4: Click: Add Exception



You are about to override how Firefox identifies this site.

Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://172.24.42.1/ Get Certificate

Certificate Status

This site attempts to identify itself with invalid information.

Unknown Identity

Certificate is not trusted, because it hasn't been verified by a recognized authority using a secure signature.

✓ Permanently store this exception

Confirm Security Exception Cancel

Figure 1.5: Click: Get certificate and confirm

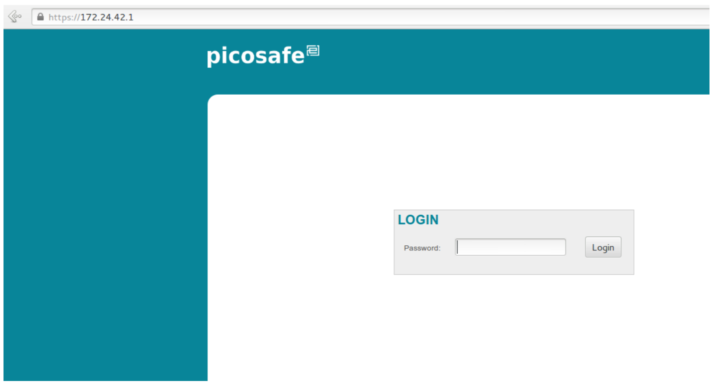


Figure 1.6: Type your disk password (default: picosafe)

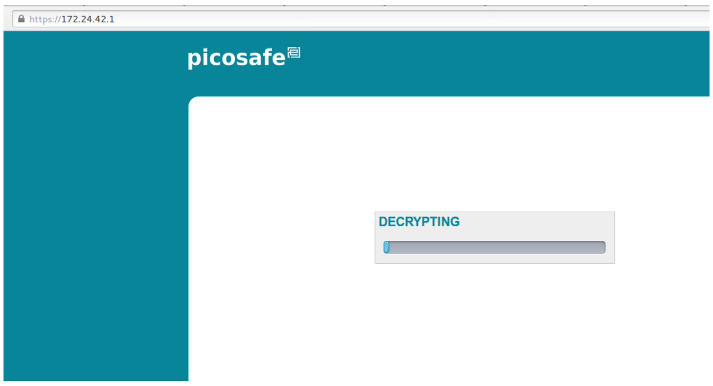
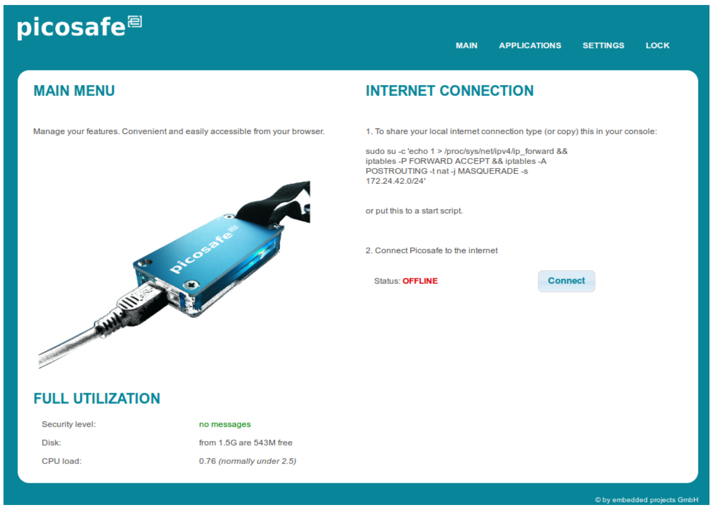


Figure 1.7: Wait while disk is decrypting



MAIN MENU

Manage your features. Convenient and easily accessible from your browser.

INTERNET CONNECTION

1. To share your local internet connection type (or copy) this in your console:

```
sudo su -c 'echo 1 > /proc/sys/net/ipv4/ip_forward &&
iptables -P FORWARD ACCEPT && iptables -A
POSTROUTING -t nat -j MASQUERADE -s
172.24.42.0/24'
```

or put this to a start script.

2. Connect Picosafe to the internet

Status: OFFLINE

FULL UTILIZATION

Security level:	no messages
Disk:	from 1.5G are 543M free
CPU load:	0.76 *(normally under 2.5)*

© by embedded projects GmbH

Figure 1.8: Welcome screen on picosafe

Chapter 2

Personalize and work with Picosafe

2.1 Connect with ssh

Now you can connect you with ssh.

User: `picosafe`
Password: `picosafe`

Start ssh connection:

```
ssh picosafe@172.24.42.1
```

Change password: `passwd`

2.2 Change boot password for cryptsetup

Default password: picosafe

You have to login via ssh and then type this commands:

- Show LUKS keys:
  `sudo cryptsetup luksDump /dev/mmcblk0p3`

  Tip: You have to type first your user password for picosafe (for sudo), the your old boot password and the two times the new password for boot.

  Now you can add a new password:

- Add a LUKS key:
  `sudo cryptsetup luksAddKey /dev/mmcblk0p3`

  And remove the old picosafe password:

- Remove LUKS key:
  `sudo cryptsetup luksKillSlot /dev/mmcblk0p3 0`

2.3 Set system clock

You can setup system clock with the web gui in menu settings, or with commandline tools like

```
sudo hwclock -set -date "2012/11/12 11:11:11"
```

and

```
sudo hwclock -hctosys
```

or online

```
sudo rdate time.fu-berlin.de
```

and then sync to hw clock `sudo hwclock -systohc`

2.4 Internet access with Linux Host

To access internet from picosafe, perform on your host (as root) this command:

```
su -c 'echo 1 > /proc/sys/net/ipv4/ip_forward && iptables -P FORWARD ACCEPT &&
iptables -A POSTROUTING -t nat -j MASQUERADE -s 172.24.42.0/24'
```

And click on the welcome main gui on connect.

2.5 Internet access with Windows

2.6 Internet access with MacOS

2.7 Shutdown Picosafe

- Disconnect
- After the automatic shutdown time (in window settings) picosafe switched of automatic
- There are two phase of shutdown
- 1. Phase LED blinks as heartbeat: That means in about 10 seconds shutdown starts
- If you will stop shutdown re-connect it fast to usb
- 2. Phase LED blinks constant: Shutdown is running

2.8 SSH Keys for further use

Picosafe is really just a completely encrypted mini server from which you can very easily but securely connect to other SSH server itself. This surface is actually more of a help, because SSH is used for Picosafe with all original tools on the command line.

With picosafe your private key never leaves the secure picosafe enviroment.

SSH tools on picosafe:

- ssh-key-gen to generate a key pair
- ssh to start ssh connections
- scp to copy files
- ssh-copy-id to copy your public key to another server or pc
- ssh-agent to remember during a session your passphrase

1. Generate first time a key pair (optional with a password)

```
ssh-key-gen
```

Additional you can protect you key pair with an passwort or so called passphrase. This means you can use the files only if you know the passpharse.

2. Add external ssh servers to your SSH config

This means you don't have to type every time ssh user@domain.de. You can put the host, user, port and so on into an config file. Then you can define a short label. After this you can start a session with ssh yourlabel. That's it.

You can do this with the web gui or directly over command line.

```
Host yourlabel
  User username_on_server
  HostName domain_or_ip
  Post port_on_server
```

Further hosts you can put also into this file.

3. Copy you public key to forgein servers

If you don't like to be ask every time you log into an server you can copy your public key on the external server. Then the server can this to authentifcate you.

```
ssh-copy-id user@domain-or-ip
```

or you can use your label from SSH config:

```
ssh-copy-id yourlabel
```

Now you have to write the last time the password. After this success you can login without a password.

4. Work with ssh-agent or remember the password for your private key during a session

If you use an password or passphrase at you keys you need every time you use the keys this password or passphrase. ssh-agent is small tool which can start an session for example for two hours. That means in this time you don't need your password or passphrase again. Before you start to work type once:

```
exec ssh-agent bash
```

Then put your keys into the hand of the agent:

```
ssh-add
```

And start to work with you ssh accounts.

5. Login to server behind servers

Sometimes a server is behind an central login server. This means you keys must be forward to the final server. You can enable this freature in you SSH config.

```
ForwardAgent yes
```

If you want to connect to a computer that is behind another, usually you have to shimmy through successively via SSH to. In the SSH config you can do this, however, define once and connect to a short machin1 ssh directly to the correct server behind other servers.

```
Host machine1
  User username
  HostName domain-or-ip-rechner1

Host machine2
  User otherusername
  ProxyCommand ssh -q machine1 nc -q0 domain-or-ip-machine2 22
```

If one wants to also now not every log times with the password you can directly with its public key on machine2.

```
ssh-copy-id machine2
```

2.9 Rescue Backup

We recommend to do the first time a complete backup (dump) of your sd card.

After put your sd card into an pc (with an reader) find out what device char you get.

```
dmesg
```

Then umount all partitions.

```
sudo umount /dev/sdX1
```

```
sudo umount /dev/sdX2
```

```
sudo umount /dev/sdX3
```

```
sudo umount /dev/sdX4
```

And final copy the complete image:

```
sudo dd if=/dev/sdX of=picosafe_backup_image.bin
```

You can recover the content to a new card with this command:

```
sudo dd if=picosafe_backup_image.bin of=/dev/sdX
```

You have to change also X with the correct char.

2.10 Update

Login with ssh (as picosafe). Then change the foler:

```
cd /opt/picosafe
```

And update with git:

```
git pull
```

Sometimes an SSL certificate error occours. This themes to be a bug of the git version. A quick fix is to export a variable:

```
export GIT_SSL_NO_VERIFY=1
```

Then do the first pull

```
git pull
```

And now enable ssl verify again.

```
export GIT_SSL_NO_VERIFY=0
```

At this time now you should be able to pull without this fix.

Chapter 3

Introductory Information

3.1 Introduction

Picosafe combines a small embedded linux device, secure booting and encrypted root filesystem, and low power consumption.

The code of the bootloader is stored encrypted and only gets decrypted at runtime. Starting from the bootloader a chain of trust is established by validating each software component. The bootloader validates kernel and initramfs and the initramfs will mount only an encrypted root filesystem. Therefore only trusted software will run on picosafe.

Figure 3.1: Picosafe USB stick.

Picosafe uses a 270 MHz LPC3143 with 192 kB internal SRAM, AES decryption unit, memory management unit, USB 2.0 and random number generator. In addition picosafe provides 32 MiB SDRAM, SD memory card both for application and data, two LEDs and an USB device.

3.2 Features

- LPC3143
  - 270 MHz, 32bit ARM926EJ-S
  - 16 kB D-cache and 16 kB I-cache
  - memory management unit (MMU)
  - 192 kB embedded SRAM
  - AES decryption unit

  - secure one-time programmable memory for AES key
  - 128 bit unique id
  - random number generator
  - High-speed USB 2.0 (OTG, Host, Device) with on-chip PHY
  - DMA controler
- 32 MB SDRAM
- SD-card for applications and data
- secure boot – established chain of trust at bootup
- linux system
- two LEDs (green and red)
- small device
- low power consumption

3.3 Security Concept

Picosafe's LPC3143 will only load encrypted code after reset. In order to run any code after reset (typically the bootloader), the code must be correctly encrypted with a 128 bit AES key. The AES key is stored in the OTP area of the LPC3143 and different for every picosafe device.

In order to establish a chain of trust, every software component must validate the next software component, i.e. the bootloader must validate kernel and initramfs, initramfs must validate the root filesystem.

Following steps are performed:

1. LPC3143 loads bootloader from SD-card
2. LPC3143 decrypts bootloader with AES key stored in OTP area
3. LPC3143 executes decrypted bootloader
4. apex bootloader loads and validates kernel and initramfs:
   - Possibility 1: Encrypting bootloader

     Kernel and initramfs are encrypted with the same AES key the LPC3143 uses to decrypt the bootloader. The apex bootloader loads kernel and initramfs from the SD-card and decrypts both. The bootloader will also check if the decrypted data contains a certain magic string to verify that the kernel is valid.

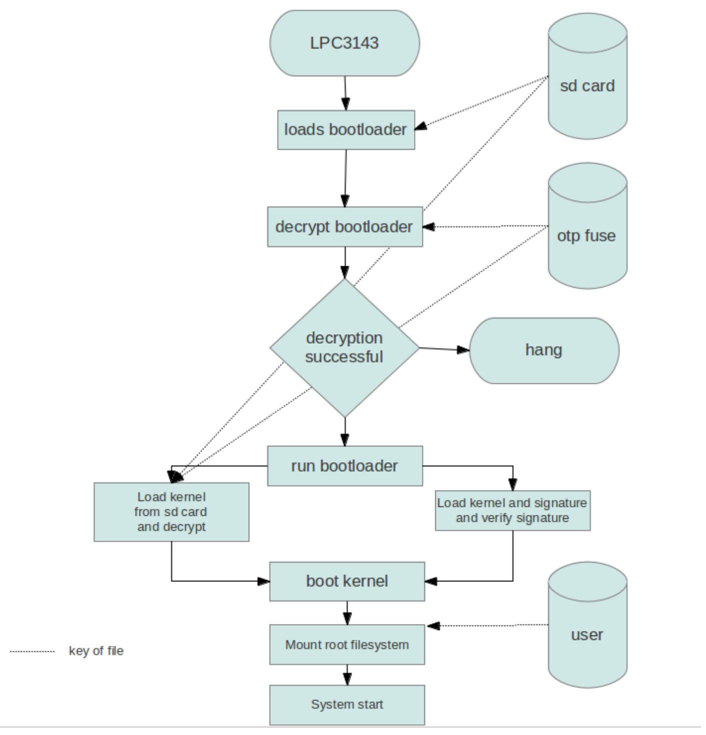


Figure 3.2: Flowchart of boot process.

- Possibility 2: Verifying signature

  Kernel and initramfs are signed. The apex bootloader contains a public key (RSA 1024bit) that can be used to verify the signature of kernel and initramfs. The apex bootloader loads kernel and initramfs, calculates the SHA-1 checksum and verifes the signature. If verification was not successful, the memory area of kernel and initramfs will be overwritten with zeroes.

5. The kernel boots up, initializes hardware and executes `init` on the initramfs. `init` sets up a network connection over USB and starts a webserver. The user can insert the password of the root-filesystem on a SSL secured webpage.
6. If the user provides the correct password, the root-filesystem can be mounted and the linux system starts.
7. The user can log in via ssh or use other services of the system such as webserver, databases etc.

The user may modify software components to meet demands. Configuring and customizing bootloader, kernel and initramfs, and root-filesystem are described in the corresponding chapters.

3.3.1 Encrypted/protected data/code

The boot process of the linux system consists of three different parts of software:

1. bootloader
2. kernel and initramfs
3. root-filesystem

The bootloader is encrypted. The AES key is stored in the OTP memory of the LPC3143 and cannot be read out.

An attacker may overwrite the bootloader on the sd-card with arbitrary data. But the LPC3143 will first decrypt the data and then execute the code. As the attacker doesn't know the AES key, the LPC3143 will execute random and probably invalid code.

For kernel and initramfs there are two possibilities:

1. Kernel and initramfs may be encrypted in the same way the bootloader is encrypted. Then the same arguments apply.
2. Kernel and initramfs are not encrypted, but the bootloader checks the signature of kernel and initramfs. If so, an attacker may read kernel and initramfs (e.g. secret strings or the configuration of the kernel), but cannot change kernel or initramfs.

The initramfs mounts the root-filesystem. To do so, the user has to insert a password to mount the root-filesystem. The root-filesystem will only open, if the supplied password is correct. So if the root-filesystems can be mounted, the files on the root-filesystem haven't been altered. (Please notice that an attacker may destroy data – e.g. by overwriting the root-filesystem or simply by breaking the SD-card into pieces.)

3.3.2 Unencrypted/unprotected data/code

Picosafe validates the integrity of bootloader, kernel and initramfs and root-filesystem. However, the SDRAM is not encrypted. An attacker may read the SDRAM at runtime or even modify e.g. kernel code in the SDRAM at run time. This is not a trivial attack, but it can be done. Also be aware of cold boot attacks! The content of the SDRAM will not magically vanish after shutdown. An attacker may read out data from the SDRAM even after poweroff!

As a precaution, you may

- Encrypt RAM:

  This is possible by creating a RAM disk with encrypted filesystem that contains a swap file. However, not all memory will be stored in the encrypted swap file, the key will be stored in plain text somewhere in the memory and the kernel code will not be encrypted at all. This will make attacks a bit harder, but it won't fix the problem.

- Clear RAM before poweroff:

  This will prevent cold boot attacks. However, you cannot ensure that you can always overwrite the RAM before poweroff. Also an attacker may still read and modify the RAM during runtime.

- Store sensible data in the internal SRAM of the LPC3143:

  For very sensible data you may use the internal SRAM of the LPC3143. However, you can only access this memory in kernel mode and you have to be very careful that your data doesn't get stored in the SDRAM as well. Also, you only have 192 kB of SRAM!

- Use emulation:

  You may adjust/write an emulator that emulates an ARM device and encrypts the external SD-RAM. This will decreace performance drastically, it will take some time to adjust/write such an emulator, but it will certainly fix the security problem.

3.4 About this document

- This document explains how the components bootloader, kernel, initramfs and root-filesystem can be compiled/generated and modified.
- This document assumes that you are using a Ubuntu 12.04 system on a 32-bit x86 system. However, most of the document should apply to all linux distrubutions.
- This document also assumes that you have basic experience in using linux and a linux shell like bash.
- If you find mistakes, please send us a mail to picosafe@embedded-projects.net.

Chapter 4

Hardware

4.1 Introduction

The picosafe hardware is small, has a low power consumption, but provides enough performance for many applications.

4.2 Features

- LPC3143:
  - 270 MHz, 32bit ARM926EJ-S
  - 16 kB D-cache and 16 kB I-cache
  - memory management unit
  - 192 kB embedded SRAM
  - AES decryption unit
  - secure one-time programmable memory for AES key
  - 128 bit unique id
  - random number generator
  - High-speed USB 2.0 (OTG, Host, Device) with on-chip PHY
  - DMA controler
- 32 MB SDRAM
- SD-card
- two LEDs (green and red)

4.3 LPC3143

The LPC3143 provides a 32-bit ARM926EJ-S with 270 MHz, 192 kB internal SRAM and USB 2.0 (OTG, host, device).

The security features include a random number generator, an AES decryption engine and a secure one-time programable (OTP) memory for AES key storage. The processor also supports running encrypted code after reset. This option is turned on by default and there is no possibility to run any code after reset without knowledgte of the correct AES key stored in the OTP area of the processor. For security reasons JTAG is completely disabled.

After reset the AES key may be read from the internal SRAM. However, it is possible to protect the AES key from being read by writing an appropriate value to the rprot (read protection) register. A full description about the OTP memory and read protection is given in the LPC314x user manual in chapter 19.

4.4 Peripherals

Picosafe contains two LEDs and a USB interface:

- A green LED that is connected with the input voltage and will be turned on when the device is supplied with power,
- and a red LED that can be turned on and off by software (GPIO3).
- USB device interface

Chapter 5

Cross Compiler

5.1 Introduction

In order to compile C/C++ programs for picosafe, you need a cross compiler. Once you have the cross compiler installed, you can compile bootloader, kernel or your own applications.

5.2 Installing

To install the cross compiler, execute these instructions:

1. Change into the directory `toolchain/` of the picosafe directory and extract the archive to your root-filesystem.

   ```
   $ cd toolchain/
   $ sudo tar xjf eldk-eglibc-i686-arm-toolchain-qte-5.2.1.tar.bz2 -C /
   ```

   This will extract the contents of the archive `eldk-eglibc-i686-arm-toolchain-qte-5.2.1.tar.bz2` to your root-filesystem. The cross compiler will be copied into the directory `/opt/eldk-5.2.1/`.

2. Source the file `picosafe.sh` to set the environment:

   ```
   $ .  /opt/eldk-5.2.1/picosafe.sh
   ```

3. Check if the installation of the cross compiler was successful:

   ```
   $ arm-linux-gnueabi-gcc -v
   [...]
   Thread model:  posix
   gcc version 4.6.4 20120303 (prerelease) (GCC)
   ```

5.3 Cross-Compiling

We will go ahead and cross compile a simple hello world program. First create a file `hello.c` with this content:

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

To compile this example:

1. Make sure, the environment is set correctly:

   ```
   $ . /opt/eldk-5.2.1/picosafe.sh
   ```

2. Compile using gcc:

   ```
   $ arm-linux-gnueabi-gcc hello.c -o hello
   ```

3. The previous command will create a file hello if successful. You can check if the output file has the correct format:

   ```
   $ file hello
   hello:  ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked
   (uses shared libs), for GNU/Linux 2.6.16, not stripped
   ```

4. If you run this file on picosafe, you should see this output:

   ```
   $ ./hello
   Hello World
   ```

Chapter 6

Apex bootloader

6.1 Introduction

The apex bootloader is the first piece of software that will be executed after a reset of the LPC3143. Therefore the apex bootloader must be encrypted correctly or the LPC3143 won't execute legal code.

Further information about apex is available at `http://elinux.org/APEX_Bootloader`.

6.2 Compiling

In order to compile the apex bootloader, you may use the script `build.sh` in the directory `bootloader/`. You must pass an action, the filename of the key (in picosafe format) and an output filename. You can create a picosafe keyfile using picosafe_genkey_aes.

```
$ ./build.sh keyfile-picosafe.key ACTION KEYFILE OUTPUTFILE
```

- `ACTION`: may be `fuse` if you want to create a fuse image to fuse the key into the LPC3143, or may be `bootloader` if you want to create a bootloader.
- `KEYFILE`: file with key in picosafe format
- `OUTPUTFILE`: filename of output

For more information, have a look at chapter 11.

6.3 Configuring and Compiling (manual)

In order to compile the apex bootloader, perform following steps:

1. First, change to the directory of the apex bootloader sources:

   ```
   $ cd apex-secure/apex
   ```

2. Make sure, the path to the cross compile is set:

   ```
   $ . /opt/eldk-5.2.1/picosafe.sh
   ```

3. Adjust the configuration of the apex bootloader to meet your demands:

   ```
   $ make menuconfig
   ```

4. Compile the apex bootloader

   ```
   $ make
   ```

After successful compilation you can find the binary at the path `src/arch-arm/rom/apex.bin`.

6.4 Important Commands

6.4.1 help

```
help [. | COMMAND]
```

Display help.

The optional command parameter will show the detailed help for commands that would match that prefix.

Examples:

```
apex> help dump      # Show help for dump command
apex> help d         # Show help for all 'd' commands
apex> help           # List the available commands
apex> help .         # Show all help topics
```

6.4.2 copy

```
copy [-s] SRC DST
```

Copy data from SRC region to DST region. Adding the -s performs full word byte swap. This is necessary when copying data stored in the opposite endian orientation from that which APEX is running. This option requires that the length be an even multiple of words. The length of the DST region is ignored.

Example: Copy the file zImage on the SD-card to the SDRAM at address 0x30008000

```
apex> copy ext2://1/zImage 0x30008000
```

6.4.3 boot

`boot [-g ADDRESS] [COMMAND_LINE]`

Boot the Linux kernel.

It reads environment variables bootaddr and cmdline as defaults for the start address and command line. The -g switch overrides the start address. The rest of the parameters, if present, override the default kernel command line.

Examples:

```
apex> boot -g 0x0008000
apex> boot console=ttyAM1 root=/dev/nfs ip=rarp
```

6.4.4 reset

Reset the system.

This will perform an immediate, hard reset of the CPU.

6.4.5 printenv

Display the environment.

The output is `KEY [*]= VALUE` where `KEY` is an environment variable name and `VALUE` is the current seting. The * denotes a variable set to the default value.

6.4.6 scopy-aes

`scopy-aes SRC`

This command will copy the file SRC to the SD-RAM at 0x30008000 and then decrypt the date using the 128-bit AES key stored in the OTP memory of the LPC3143.

You can encrypt the compiled apex bootloader using the tool `picosafe_aes`. This tool is described in the chapter tools.

Example:

```
apex> scopy-aes ext2://1/zImage.crypt
```

6.4.7 scopy-sig

`scopy-sig SRC SIG`

This command will first read the signature from file `SIG` to ram, then copy the file `SRC` to the SDRAM at 0x30008000 and then verify the signature of `SRC`. If the signature is not valid, the memory of the SDRAM beginning at 0x30008000 will be filled with zeroes.

The apex bootloader will need the correct public key to verify the signature: You can generate a RSA key pair using the tool `picosafe_genkey`. `picosafe_genkey` will also create a file pubkey.h. Copy the file pubkey.h into the directory include/ of the apex/ directory and recompile the bootloader.

You can create a signature using the tool `picosafe_sign`.

The usage of these tools are described in the chapter tools.

Example:

```
apex> scopy-sig ext2://1/zImage ext2://1/zImage.sig
```

6.4.8 fuses

`fuses ACTION [PARAMETERS]`

This command allows you to read and write fuses of the OTP section of the LPC3143.

`ACTION` may be

- `blow_aes_key AES_KEY`

  Write the aes key `AES_KEY` to the poly fuses of the LPC3143. The AES key is 128bit (16 bytes). `AES_KEY` must start start with the prefix `0x`, afterwards every byte is encoded as two hexadecimal characters.

  Example: Write the key `ABCDEFGHIJKLMNOP`

  ```
   fuse blow_aes_key 0x4142434445464748494A4B4C4D4E4F50
  ```

- `aes_enable`

  Enable AES encrypted boot. The command will blow the fuse bit 504. This will indicate that the AES key in fuses 128:255 is valid. Write the AES key before enabling AES encrypted boot. After enabling AES boot, you can only boot encrypted code. (See `blow_aes_key`)

- `read`

  Read out the fuses for `OTP_DATA_0` to `OTP_DATA_15` (bit 0 to bit 511).

  The output will look like:

```
OTP_DATA_00: 0xXXXXXXXX
OTP_DATA_01: 0xXXXXXXXX
...
OTP_DATA_15: 0xXXXXXXXX
```

- `set_security_level SECURITY_LEVEL`

  Set the security level of the LPC3143.

  `SECURITY_LEVEL` is may be

  0: nothing is protected (Level 0)

  1: password protected. In this level, JTAG can be enabled by software after passwort sequence (depends on customer application) by setting the sticky bit `JTAG_EN` in `OTP_con` register. (Level 1)

  2: In this level, JTAG access can be enabled using special test equipment. Used by NXP for Returned Material Analysis only. (Level 2)

  3: JTAG is completley disabled and hence the chip is virtually locked. (Level 3)

- `blow FUSEBIT`

  This command will blow the fuse bit `FUSEBIT` and set it to 1.

6.5 Customizing

You may also write your own apex commands.

For a simple example we will create a command test:

1. First, we create a file `cmd-test.c` in the directory `src/apex`

2. We save the code of our command in the file `src/apex/cmd-test.c`:

```
#include <config.h>
#include <linux/string.h>
#include <apex.h>
#include <command.h>
#include <driver.h>
#include <error.h>

int cmd_test(int argc, const char** argv) {
  printf("Hello test\n");
  return 0;
}

static __command struct command_d c_info = {
  .command = "test",
  .func = cmd_test,
  COMMAND_DESCRIPTION("Test command")
  COMMAND_HELP("test"
"  this command will output the String \"Hello test\"\n")
};
```

3. Add command to src/apex/Kconfig
4. Add object to src/apex/Makefile
5. Check configuration and recompile the bootloader:

```
$ make menuconfig
$ make
```

Chapter 7

initramfs

7.1 Introduction

An initramfs (initial ram filesystem) is a compressed archive that contains all needed files and programs to boot a linux system. The kernel mounts the content of the initramfs to the root-directory and starts a program (typically `/sbin/init` or `/init`) that mounts the real root-filesystem and starts the linux system.

In order to start up fast the initramfs should be small, therefore tools like busybox are used.

7.2 Creating initramfs

In order to create a initramfs, you may use the script `geninitramfs.sh`. The script performs following steps:

1. Checking, if the cross compiler is installed correctly.
2. Creating the directory structure for the initramfs.
3. Creating device nodes like `/dev/null` or `/dev/console`.
4. Copying busybox to the initramfs. The script will compile busybox first if needed.
5. Installing all Debian packages in the directory `packages/`.
6. Copying kernel modules. The script will compile the kernel modules first if needed.
7. Copying the content of the directory `initramfs-root/` to the initramfs.

To create the initramfs, change to the directory `initramfs/` and execute the script `geninitramfs.sh`. This script will need root priviledges in order to create device nodes.

```
$ cd initramfs/
$ ./geninitramfs.sh
Deleting old initramfs directory...
Creating directory structure...
Creating device nodes...
Compiling busybox not needed (skipped).
Creating important symlinks...
Copying busybox to initramfs...
Installing packages:
cryptsetup-udeb_1.1.3-4squeeze2_armel.udeb...
libc6-udeb_2.11.3-4_armel.udeb...
libdevmapper1.02.1-udeb_1.02.48-5_armel.udeb...
libgcc1_4.4.5-8_armel.deb...
libncurses5_5.7+20100313-5_armel.deb...
libpopt0-udeb_1.16-1_armel.udeb...
libreadline6_6.1-3_armel.deb...
libuuid1-udeb_2.17.2-9_armel.udeb...
lua5.1_5.1.4-5_armel.deb...
readline-common_6.1-3_all.deb...
Installing kernel modules...
Copying files into initramfs...
./init...
./var/www/index.html...
./var/www/cgi-bin/password.lua...
./etc/httpd.conf...
./etc/udhcpd.conf...
Done.
```

The initramfs will be created in the directory `output/`.

7.3 Configuring initramfs

You may change the initramfs to meet your demands. This may include adding or removing kernel modules, installing further programs or changing the `/init` program.

7.3.1 Adding and removing kernel modules

To add or remove kernel modules, open the script `geninitramfs.sh` and adjust the variable `KERNELMODULES`. This variable is a array that stores all kernel modules that should be copied to the initramfs.

7.3.2 Busybox

Busybox provides most of the standard unix tools like `cp` or `ln`, including a shell. You may adjust the configuration of busybox and add or remove features.

To configure busybox, change to the directory `initramfs/busybox` and run `make menuconfig`. If you want to use the default configuration, copy the file `config_busybox` to the current directory. Make sure the path for the cross compiler is set correctly.

```
$ . /opt/eldk-5.2.1/picosafe.sh
$ cd initramfs/busybox
$ cp ../config_busybox . # if you whish to use the default configuration
$ make menuconfig
```

7.3.3 Adding and removing programs

If you wish to add a program or librariy to the initramfs, copy the Debian package (for ARM) in the directory `initramfs/packages`. The script `initramfs.sh` will install the Debian package to your initramfs. Please note that no dependencies are resolved. So, if you want to install a package A that depends on package B, make sure to copy the Debian packages A and B in the directory `initramfs/packages`.

If you don't need a certain program or library on the initramfs, just remove the packages in the directory `initramfs/packages`.

7.3.4 Adding files to the initramfs

You may need to add your own files to the initramfs. These files may be configuration files or your own programs.

To add your own files to the initramfs, copy the files to the directory initramfs-root. The script gen-intramfs.sh will copy all files in this directory to the root-directory of the initramfs. If the file already exists on the initramfs, the file will be overwritten.

7.3.5 /init

`/init` is the program that will be executed by the kernel with pid 1. You may change this script to meet your demands. You may use a script or a binary executable.

The default `/init` script will perform following steps:

1. Creating symlinks to busybox applets.
2. Mounting `/proc`, `/sys` and the SD-card.
3. Setting up a network connection over USB and starting a dhcp server.
4. Starting a webserver. This webserver will provide a page with a password field. The user then must insert the password for the encrypted root filesystem.

5. If the user provided the password, the encrypted root-directory is mounted and the linux system is started.

7.4 Hiawatha webserver

Hiawatha is a lightweight webserver with SSL and CGI support.

7.4.1 Compile (automatic)

To compile hiawatha webserver for picosafe, you may use the `build.sh` script:

1. Make sure, you have at least version 2.8.4 of `cmake` installed.
2. Change to the directory `initramfs/hiawatha/build/`

   ```
   cd initramfs/hiawatha/build/
   ```

3. execute the script `build.sh`

   ```
   ./build.sh
   ```

This will compile the hiawatha webserver and also copy the hiawatha binary and the libpolarssl to the initramfs directory.

7.4.2 Compile (manual)

To compile hiawatha webserver for picosafe, perform following steps:

1. Make sure, you have at least version 2.8.4 of `cmake` installed.
2. Download the latest source from `http://www.hiawatha-webserver.org/`.
3. Set the environment to use the cross compiler:

   ```
   . /opt/eldk-5.2.1/picosafe.sh
   ```

4. Extract the source ball and change to the new directory:

   ```
   tar xzvf hiawatha-8.6.tar.gz; cd hiawatha-8.6/
   ```

5. Create a directory `build/` and change to it:

   ```
   mkdir build; cd build/
   ```

6. Create a file `crosscompile.cmake` with this content:

```
SET(CMAKE_SYSTEM_NAME Linux)  # Tell CMake we're cross-compiling
include(CMakeForceCompiler)
# Prefix detection only works with compiler id "GNU"
# CMake will look for prefixed g++, cpp, ld, etc. automatically
CMAKE_FORCE_C_COMPILER(arm-linux-gnueabi-gcc GNU)
```

7. Run `cmake` to create the makefiles:

   ```
   cmake ..  -DCMAKE_TOOLCHAIN_FILE=./crosscompile.cmake -DENABLE_XSLT=off
   ```

8. compile

   ```
   make
   ```

7.4.3 Configuration

If you want to use SSL, create a certificate and put it in the hiawatha configuration directory:

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out hiawatha.pem
```

Chapter 8

Linux Kernel

8.1 Introduction

The linux kernel provides abstraction layers between hardware and programs. It performs essential sevices like memory management and CPU scheduling, and provides APIs for accessing hardware.

8.2 Configuring and Compiling

To configure and compile the kernel, change to the directory `kernel/linux-2.6.33-lpc313x` and run `make menuconfig` to change the configuration, and then run make zImage and make modules. Make sure the path to your cross compile is set correctly. If you want to use a initramfs, make sure that the directory with the initramfs is set up correctly. In general, you need to run geninitramfs.sh before compiling the kernel. See the chapter about initramfs.

```
$ . /opt/eldk-5.2.1/picosafe.sh
$ cd kernel/linux-2.6.33-lpc313x/
$ make menuconfig
$ make zImage
$ make modules
```

This will compile the linux kernel (`make zImage`) and the kernel modules (`make modules`). Your compiled kernel will be stored in the file `arch/arm/boot/zImage`.

8.3 Hardware Random Number Generator

It is possible to access the hardware random number generator of the LPC3143 from userspace. To do so, you will need the driver lpc314x-rng (Device Drivers → Character Devices). If the driver is compiled as a module, make sure you have loaded the module:

```
$ modprobe lpc314x-rng
```

To read data from the random number generator you just need to read the file `/dev/hwrandom`. If this device file does not exist, you may create it using `mknod`:

```
$ mknod /dev/hwrandom c 10 183
```

8.4 LED

There is a driver available to control picosafe's LED. The driver is called `leds-picosafe` (Device Drivers → LED Support).

If the driver is loaded, you can access the LED through the `/sys` filesystem.

Examples:

- Turn LED on:

  ```
  echo 1 > /sys/class/leds/picosafe::blue/brightness
  ```

- Turn LED off:

  ```
  echo 0 > /sys/class/leds/picosafe::blue/brightness
  ```

- Use heartbeat trigger:

  ```
  echo heartbeat > /sys/class/leds/picosafe::blue/trigger
  ```

- Show available triggers:

  ```
  cat /sys/class/leds/picosafe::blue/trigger
  ```

- Disable any trigger:

  ```
  echo none > /sys/class/leds/picosafe::blue/trigger
  ```

Chapter 9

Picosafe tools

9.1 Introduction

This chapter describes tools you need to encrypt and sign the kernel, so picosafe will boot it.

9.2 Compiling

To compile the programs change to the directory `tools` and execute `make`:

```
$ cd tools
$ make
```

This will compile the tools `picosafe_aes` `picosafe_genkey_rsa`, `picosafe_genkey_aes` and `picosafe_sign`.

If you wish to install these tools, run

```
$ sudo make install
```

to copy the files to `/usr/bin`.

9.3 picosafe_aes

```
picosafe_aes:  [OPTIONS] FILE
```

This program will encrypt/decrypt FILE.

Options may be:

- `-k FILENAME`: use key stored in FILENAME (if omitted, will try key.txt)
- `-o FILENAME`: write output to FILENAME (if omitted, FILE will be overwritten)
- `-d`: don't encrypt, but decrypt FILE
- `-h`: print help

Example: Encrypt file `/mnt/zImage` and write it to `/mnt/zImage.crypt`:

```
$ picosafe_aes -d /mnt/zImage.crypt /mnt/zImage
```

9.4 picosafe_genkey_aes

```
picosafe_genkey_aes:  [OPTIONS] SUFFIX
```

This program will create AES key files for lpc-tools and picosafe tools. The files created are `FILE_-picosafe.key` and `FILE_lpc.key`.

Options may be:

- `-h`: print help

Example: Create a random key and store it in the files `key_picosafe.key` and `key_lpc.key`

```
$ picosafe_genkey_aes key
```

9.5 picosafe_genkey_rsa

```
picosafe_genkey_rsa [outputdir]
```

This program will create a RSA 1024bit key pair. The public and private keys are written to the files `picosafe_pub.txt`, `pubkey.h` (public key) and `picosafe_priv.txt` (private key).

If no output directory is given, the current directory will be used.

The private key (`picosafe_priv.txt`) can be used to sign files using `picosafe_sign`.

The public key can be used to verify signatures. The file `pubkey.h` can be used by the apex bootloader to verify the signature of a kernel.

Please read the picosafe manual on apex for further information.

9.6 picosafe_convert

This program converts keys saved in picosafe format to the LPC format.

Options may be:

- `-h, -help`: show help message and exit
- `-f FILENAME, -file=FILENAME`: input key file in picosafe format
- `-o OUTPUT, -output=OUTPUT`: output key file in lpc format (default: stdout)
- `-n, -newline`: add trailing newline

9.7 picosafe_sign

`picosafe_sign: [OPTIONS] FILE`

This program creates a signature for file `FILE`.

The private key must be generated by `picosafe_genkey`. If the flag `-p` is not given the file `picosafe_-priv.txt` will be used as private key.

The program will store the signature to the file `FILE.sig`.

Options:

- `-p PRIVATEKEY` path to private key
- `-h` show this help

Chapter 10

Creating SD-card

10.1 Introduction

This chapter describes how to create a SD-card that will boot on your picosafe.

10.2 Creating partition table and LUKS container

If you just want to create a SD-card with a running linux system, you may use the script `genrootfs.sh` in the directory `rootfs/`. This script will perform following steps:

1. Filling the SD-card with random data.
2. Creating the partition table and the filesystems.
3. Creating a LUKS container with a ext3 filesystem.
4. Copying the linux root filesystem to the LUKS container.
5. Copying the linux kernel modules to the LUKS container.
6. Copying the linux kernel to the SD card.
7. Copying the apex bootloader to the SD card.

To create a SD-card, run:

```
$ cd rootfs
$ sudo ./genrootfs.sh /dev/sdX keyXX_picosafe.key
```

PEM_FILE

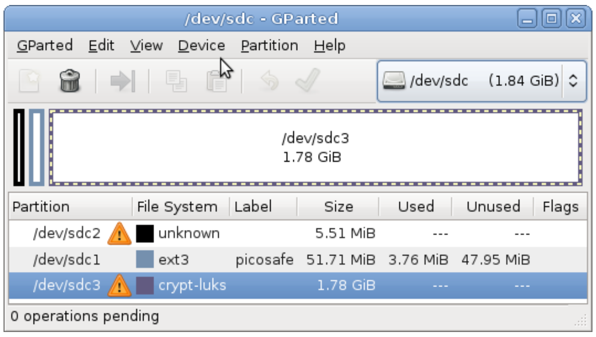
where `/dev/sdX` is the block device of your SD-card. You may specify a `PEM_FILE` that will be used for the webserver. If omitted, the file will be generated.

The default password for the LUKS container is `picosafe`.

10.3 Partition table

This section gives aditional information on the partition table and how to create it yourself. So you usually can just skip this section.

The SD-card must contain three primary partitions: A bootit (id `0xDF`) partition with size of a few megabytes that contains the apex bootloader, a partition formated with ext3 (id `0x83`) that contains the kernel, and a partition that contains the LUKS container with the root filesystem. The bootit partition must be physically the first partition on the SD-card, i.e. it must start after partition table. It is recommended that the bootit partition is (logically) the second primary partition, and the ext3 partition with the kernel is the first primary partition. The bootit partition should have a size of few megabytes, the ext3 partition should use about 50 megabytes. The last partition for the LUKS container should use all the space left on the SD-card.



Partition	File System	Label	Size	Used	Unused	Flags
/dev/sdc2	unknown		5.51 MiB	---	---	
/dev/sdc1	ext3	picosafe	51.71 MiB	3.76 MiB	47.95 MiB	
/dev/sdc3	crypt-luks		1.78 GiB	---	---	

0 operations pending

Figure 10.1: Partition table of SD-card shown by gparted.

If you want to create the partition table yourself, you may use your favourite tool or you may use the script `partition.sh` in the directory `rootfs`. This script will use fdisk to create the partition table and format the ext3 filesystem. You only need to pass the blockdevice of the SD-card to the script. As you are writing to a block device, you will need root privilidges. Also make sure that your are giving the right block device as argument to the script. Be carefull, as this may destroy any data on the block device! Also make sure that there is no partition of the SD-card mounted when executing the script.

```
$ cd rootfs
$ sudo ./partition.sh /dev/sdX
```

10.4 Copying kernel

Mount the ext3 filesystem of the SD-card and copy the encrypted kernel to it (see chapter Compiling kernel).

```
$ sudo mount /dev/sdX1 /mnt
$ sudo picosafe_aes -k key.txt -o /mnt/zImage.crypt zImage
```

10.5 Copying apex bootloader

In order to copy the apex bootloader to the bootit partition of the SD-card, you may use the `dd` command. Be sure to pass the correct block device as `dd` will overwrite any existing data.

```
$ sudo dd if=../bootloader/apex.rom of=/dev/sdX2
```

Chapter 11

Production

11.1 Introduction

This chapter describes the production of a picosafe device.

11.2 Writing AES key (automatic)

1. Generate a key

   ```
   $ picosafe_genkey_aes keyfilename
   ```

2. Change to the directory `bootloader/` and run

   ```
   ./build.sh -a fuse -k keyfilename_picosafe.key -o apex-fuses-fuse.dfu.rom
   ```

   This will create the file `apex-fuses-dfu.rom`.

3. Start the dfu bootloader once

   ```
   sudo dfu-util -R -t 2048 -D apex-fuses-dfu.rom
   ```

11.3 Writing AES key (manual)

1. Generate keys:

   ```
   $ picosafe_genkey_aes keyfilename
   ```

2. Connect picosafe to USB and wait for a device PHILIPPS (check `lsusb`).

3. Connect USART cable.

4. Connect to USART:

   ```
   $ picocom -b 115200 /dev/ttyUSB0
   ```

5. Start apex bootloader using `dfu-boot`

   ```
   $ sudo dfu-util -R -t 2048 -D bootloader/apex-dfu.rom
   ```

6. Blow aes key:

   ```
   apex> fuse blow_aes_key AES_KEY
   ```

7. Enable encrypted boot:

   ```
   apex> fuses aes_enable
   ```

8. Set security level:

   ```
   apex> fuses set_security_level 3
   ```

9. Disable DFU boot:

   ```
   apex> fuses blow 502
   ```

11.4 Creating a new root-filesystem

This section will show you, how to save the filesystem of a picosafe stick. You can then use the root-filesystem to create new picosafe sticks.

1. Open the picosafe device, extract the SD card and insert it into your pc.-
2. Mount the encrypted LUKS container.
3. Copy the root-filesystem to a local directory.
4. Change to the directory and become root.

   ```
   $ cp -a /media/b707efce-7235-420f-b664-d92679c79b81/* rootfs/
   ```

5. Remove the swap file. The swap file be created automatically.

   ```
   $ rm swapfile
   ```

6. Delete private files in the home directories `root/` and `/home/picosafe`:

   ```
   > root/.bash_history
   > root/.viminfo
   rm -r root/.subversion
   rm -r root/.ssh
   > /home/picosafe/.bash_history
   > /home/picosafe/.viminfo
   rm -r /home/picosafe/.subversion
   rm -r /home/picosafe/.ssh
   ```

   Make sure to delete any private files and not only the listed ones.

7. Delte `.svn/` directories:

   ```
   $ rm -r ‘find -name .svn -type d‘
   ```

8. Create tarball:

   ```
   $ tar -preserve-permissions -preserve-order -numeric-owner -czf ../picosafe_-
   rootfs.tar.gz .
   ```