University Bordeaux I
Master I Cryptologie et Sécurité Informatique

Project supervised by Mr Fleury

Linux4Nano's knowledge about the Apple iPod Nano 2G

Project Web Site: http://home.gna.org/linux4nano/

An original document written by:
Jean-Christophe Delaunay and Clément Sudron

Bordeaux, April 2010

Contents

I Brief history of Apple iPods **4**

1 An history on the iPod: features and evolution **5**
1.1 Summary . . . 5
1.2 iPod "Classic" . . . 6
1.3 iPod Mini/Nano . . . 6
1.4 iPod Shuffle . . . 6
1.5 iPod Touch . . . 7
1.6 Complements . . . 7

2 Other projects **8**

II Firmware study **9**

3 Description of the firmware **11**
3.1 iPod overview . . . 11
3.2 Get the firmware from the iPod / Put a firmware on the iPod . . . 11
3.3 Firmware partition . . . 14
3.4 The files osos.fw, aupd.fw and rsrc.fw . . . 15

4 Encryption **19**
4.1 Encryption on previous iPods . . . 19
4.2 Testing if it is a "basic" encryption . . . 19
4.3 Determination of the kind of algorithm used . . . 21
4.4 Current assumptions . . . 22
4.5 Conclusion . . . 23

5 "Notes" buffer overflow **24**
5.1 Noticing . . . 24
5.2 Exploiting, getting execution . . . 25

III Hardware study **27**

6 Components **29**
6.1 Components list . . . 29
6.2 ARM 940T Survey . . . 31

7 JTAG research **33**
7.1 JTAG definition, and its usage . . . 33
7.2 JTAG location . . . 33
7.3 JTAG cache dumps . . . 34
7.4 Conclusion . . . 34

IV iLoader and RockBox firmware **35**

8 Towards iLoader **37**
8.1 Introduction . . . 37

8.2 The "Notes" overflow exploit . . . 37
8.3 The encryption / decryption process . . . 37
8.4 iBugger . . . 38

9 Versions **39**
9.1 NAND-based iLoader . . . 39
9.2 NOR-based iLoader . . . 39

10 Tutorial about iLoader **41**
10.1 iLoader Installation . . . 41
10.2 Installing RockBox . . . 42
10.3 Making Apple firmware work . . . 43
10.4 Updates . . . 43
10.5 Uninstallation . . . 43
10.6 Known issues . . . 43

V Appendices **45**

A Modes and keys combinations **46**

B iPods hardware specifications **49**

Credits **60**

Preface

On September 2006, Apple introduced a second generation of iPod Nano. This new line of products brought quite a lot of innovations and improvements such as a brighter LCD display, an extended battery life and was available in 2, 4 and 8GB models.
Despite the fact that only one year had elapsed since the introduction of the first generation of iPod Nanos, the second generation was not only an upgrade compared to first generation but a total rebuild of it. The hardware had totally changed and some extra protections had been added to prevent the firmware from being corrupted. For example, several low level processes, such as the boot process and firmware updates, had been rethought, made more reliable and more difficult to hack.
On the other hand, higher level processes such as the operating systems and software applications remained almost untouched, the only sequel of the change was a complete recompilation for the new ARM processor in use on this new version of the iPod Nano.
In January 2007, the Linux4Nano project[1] started as a student project at the ENSEIRB (engineer school) in Bordeaux (France). The goal of the project was to provide a complete analysis of the Apple iPod Nano 2G (second generation) boot process in order to port an ARM Linux kernel on it. The student project ended in May 2007 and was unfortunately not successful in providing enough information to achieve this goal.
We are now in 2010 and we were given the role to make a report concerning what has been done, as our *Master I* project. Today, the original goal has far evolved, and most of efforts are focused on finding a way to understand all aspects of the security of each iPods' generation, in order to port RockBox[2], a community project, and other third-part projects on them. Nevertheless, this report will focus on the iPod nano 2G, as it was our first goal.
In a few words, the aim of this Master project report is to sum up all the knowledge which was accumulated during the past three years. It is supposed to embrace all you need to know about the iPod Nano 2G to help for its reverse engineering. We will try to explain as clearly as possible the hypothesis which have been made, and the facts which have been found.

This report is divided into five parts:

- a short history of iPods (part I)
- the firmware (part II): its structure, encryption and weaknesses
- the hardware (part III): the components and research of JTAG
- how it is now possible to run RockBox[2] on the Nano 2G (part IV)
- appendices (part V): a copy of two important iPodLinux[3] wiki page

[1] http://home.gna.org/linux4nano/
[2] http://www.rockbox.org/
[3] http://www.ipodlinux.org/

Part I

Brief history of Apple iPods

1 An history on the iPod: features and evolution

Preliminary remark: by now, we may write iPod Nano or IN2G instead of "iPod Nano Second Generation"; for other models, we will always specify.

1.1 Summary

On the graphic 1.1, you can see the evolution of iPods and their capacity. Please excuse us, this graphic does not show the last versions (Classic 3G, Nano 5G, Shuffle 3G and Touch 3G), which appeared on september 2009.
Remark: Apple calls the iPod Classic that they brougth out in september 2008 the "Classic 2G", and the one in september 2009 the "Classic 3G"... even if there are not really differences between these models.

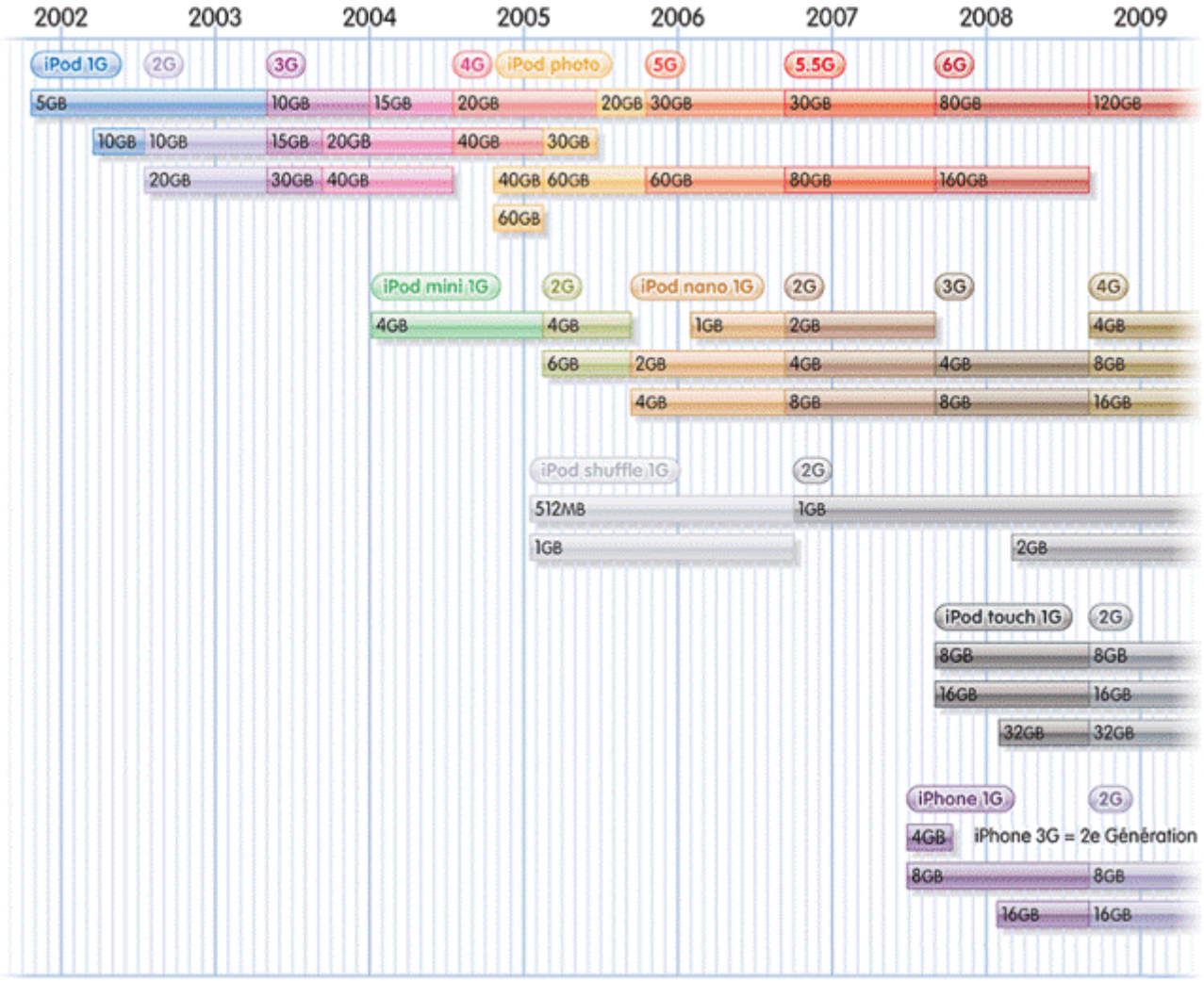

Fig. 1.1: iPod timeline, from the first release to 2009

The following facts do not deal with hardware specificities of each model (if you want to learn more about this, have a look to appendix B). It is only "notes" about the new things each model brougth in relations to the previous ones.

1.2 iPod "Classic"

Fig. 1.2: iPod "Classic"

- 1G: first release
- 2G: compatibility with Windows (2000/XP), touch wheel
- 3G: USB2 connector, dock connector, photo card reader
- 4G: click wheel, firmware is ciphered[1]
- Photo/Color: color screen
- 5G (replace iPod Color): USB only, can read videos
- 5.5G: can run downloaded games
- 6G ("Classic"): Cover Flow, new encryption[2]

1.3 iPod Mini/Nano

Fig. 1.3: iPod Nano 2G

- Mini 1G: smaller than classic iPods, click wheel, storage on microdrive
- Mini 2G: no more FireWire link
- Nano 1G (replace iPod Mini): color screen, USB only, storage on flash memory
- Nano 2G: new encryption[2], DFU mode, access to the NAND flash with a FTL software
- Nano 3G: Cover Flow, can read videos, can run downloaded games
- Nano 4G: accelerometer
- Nano 5G: CMOS camera, Voice Over Commands

1.4 iPod Shuffle

Fig. 1.4: iPod Shuffle 3G

- Shuffle 1G: very small, no screen, storage on flash memory
- Shuffle 2G: no real changes (new design...)
- Shuffle 3G: Voice Over Commands

[1] one of the three images inside the firmware is encrypted (with RC4); see part 4.1.1 of this report
[2] two of the three images are now encrypted (with AES ?); see part 4.1.2 (and following) of this report

1.5 iPod Touch

- Touch 1G: touch screen, Cover Flow, access to the Internet, storage on flash memory
- Touch 2G: integrated speaker
- Touch 3G: CPU and GPU are faster than 2G's ones, CMOS camera

Fig. 1.5: iPod Touch 3G

1.6 Complements

For the last models :
- audio formats supported, according to Apple's website: AAC (8 to 320 Kbps), Protected AAC (from iTunes Store), HE-AAC, MP3 (8 to 320 Kbps), MP3 VBR, Audible (formats 2, 3, 4, Audible Enhanced Audio, AAX, AAX+), Apple Lossless, AIFF, and WAV (with the exception of iPod Touch: HE-AAC, Audible Enhanced Audio, AAX and AAX+ aren't supported)
- video formats supported (for iPods which can read videos of course): MPEG-4 and H.264 (MPEG-4 AVC).

Each iPod is formatted in FAT-32 (Windows/Linux) or HFS+ (Mac). One exception: iPod Shuffle is formatted in FAT-32 whatever your OS is.

If you want more precise fact sheets about a model (CPU, RAM, storage,...), we have copied the iPodLinux page about hardware specifications of the different iPod generations[1]. It is available in the appendix B. You can also read technical specifications on Apple's website[2]: click on a model, and then on "Tech Specs" (top right of the page).

[1] http://ipodlinux.org/wiki/Generations
[2] http://www.apple.com/itunes/

2 Other projects

Linux4Nano is not the only project trying to run a third-part firmware on iPods. We can mainly quote iPodLinux[1] and RockBox[2].
You can find which models can run their firmwares on them, on the "Project status" page[3].

[1] http://ipodlinux.org/
[2] http://www.rockbox.org/
[3] http://ipodlinux.org/wiki/Project_Status and http://www.rockbox.org/wiki/IpodPort#Rockbox_Status

Part II

Firmware study

Table of Contents

3 Description of the firmware **11**
- 3.1 iPod overview . . . 11
- 3.2 Get the firmware from the iPod / Put a firmware on the iPod . . . 11
  - 3.2.1 Get the firmware from the Internet . . . 11
  - 3.2.2 Some preliminary explanations . . . 11
  - 3.2.3 Under Linux . . . 12
  - 3.2.4 Under Windows . . . 13
- 3.3 Firmware partition . . . 14
  - 3.3.1 Firmware structure . . . 14
  - 3.3.2 Volume header . . . 14
  - 3.3.3 Directory entries . . . 14
- 3.4 The files osos.fw, aupd.fw and rsrc.fw . . . 15
  - 3.4.1 Extraction . . . 15
  - 3.4.2 Structure and differences between successive versions . . . 16
  - 3.4.3 Modifying the content of the firmware . . . 17

4 Encryption **19**
- 4.1 Encryption on previous iPods . . . 19
  - 4.1.1 Encryption on iPod 4G, 5G, and 5.5G . . . 19
  - 4.1.2 Comparison with the IN2G . . . 19
- 4.2 Testing if it is a "basic" encryption . . . 19
  - 4.2.1 Entropy . . . 19
  - 4.2.2 Monoalphabetic substitution . . . 20
  - 4.2.3 Polyalphabetic substitution . . . 20
  - 4.2.4 Substitution by 4 bytes blocks . . . 21
  - 4.2.5 Pattern research . . . 21
  - 4.2.6 Compression . . . 21
- 4.3 Determination of the kind of algorithm used . . . 21
- 4.4 Current assumptions . . . 22
  - 4.4.1 Encryption algorithm . . . 22
  - 4.4.2 Keys and Initialization Vector location . . . 23
- 4.5 Conclusion . . . 23

5 "Notes" buffer overflow **24**
- 5.1 Noticing . . . 24
  - 5.1.1 Buffer overflow noticing . . . 24
  - 5.1.2 Creation of a working overflow file . . . 24
  - 5.1.3 Link overflow . . . 25
- 5.2 Exploiting, getting execution . . . 25
  - 5.2.1 Dumping memories . . . 25
  - 5.2.2 UART . . . 25
  - 5.2.3 USB . . . 26

3 Description of the firmware

Preliminary remark: in case of problems due to a modification of the firmware, there are some keys combinations to reboot, switch to disk mode, or diagnostic mode (see appendix A).

3.1 iPod overview

With the tool `fdisk`[1], you can display the partition table of a device. We did that on the IN2G, and figure 3.1 shows the result.
As you can see, there are two partitions : the first one (94 MB) contains the firmware, and the second one contains all the data.

```
root@[redacted]:/home/[redacted]# fdisk -l /dev/sdc
Note: sector size is 2048 (not 512)

Disk /dev/sdc: 2030 MB, 2030043136 bytes
179 heads, 52 sectors/track, 106 cylinders
Units = cylinders of 9308 * 2048 = 19062784 bytes
Disk identifier: 0x20202020

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1           6       96264    0  Empty
Partition 1 has different physical/logical beginnings (non-Linux?):
     phys=(0, 1, 1) logical=(0, 1, 12)
Partition 1 has different physical/logical endings:
     phys=(2, 254, 63) logical=(5, 31, 43)
Partition 1 does not end on cylinder boundary.
/dev/sdc2               6         107     1886072    b  W95 FAT32
Partition 2 has different physical/logical beginnings (non-Linux?):
     phys=(3, 0, 1) logical=(5, 31, 44)
Partition 2 has different physical/logical endings:
     phys=(61, 178, 52) logical=(106, 88, 7)
```

Fig. 3.1: IN2G partition table

3.2 Get the firmware from the iPod / Put a firmware on the iPod

3.2.1 Get the firmware from the Internet

You can get the last version firmwares (for all iPods) from the website of Felix 'fxb' Bruns[2].
After download, change the .ipsw into .zip, extract it, and you will get a file named "Firmware-w.x.y.z", which is the one you can write on the iPod.
There is another file, which name is *manifest.plist*: it is an XML file that gives basic informations about the firmware (probably for iTunes).

3.2.2 Some preliminary explanations

The IN2G flash only allows reading / writing by 2048 bytes long blocks.

[1] http://www.fdisk.com/fdisk/
[2] http://www.felixbruns.de/iPod/firmware/

If you want to dump it, you may need to specify an option: `count`, which value is a number of blocks.
As written above, a block size is 2048 bytes, but concerning *dd*, the default size is 512 bytes: if you want to use `count`, you have to add `bs=2048` to the options.
The `count` option represents the number of blocks `dd` will dump, and this number depends on the version of the firmware which is on your iPod. Table 3.1 shows, for each version of the IN2G firmware, its size in bytes and in blocks. That way, if you want to dump a version 19.1.1.2 of the firmware for example, you will have to use options `bs=2048 count=11806`.
To know the version of the firmware on your iPod use iTunes, or if you can't, just dump the whole firmware with `count=11860`.
In the other direction (ie if you want to write a firmware on the iPod), you do not need this option: `dd` will copy the whole file, and then stop.

Version	Sizes (bytes)	Size (blocks)
19.1.1.0	24178688	11806
19.1.1.1	24289280	11860
19.1.1.2	24178688	11806
19.1.1.3	24164352	11799

Tab. 3.1: IN2G firmwares sizes

Remark: there is a 29.1.1.3 version of the firmware. An hash of the file reveals that it is exactly the same as the 19.1.1.3 version.

3.2.3 Under Linux

Pay attention : all commands have to be executed in `root` mode, ie : equivalent command is `sudo`.

CAUTION :
You have to know the exact name of the partition bound at your iPod. To find it, you can use the command `dmesg` which displays the Kernel's buffer memory, or the command `lsscsi` which displays all scsi peripherals connected to the computer (this command may not be included by default).
All the commands below are written assuming that the iPod is recognized as sdc (/dev/sdc).

3.2.3.1 MBR

Commands to save and restore the MBR and the partition table.

If you want to save it, type :
```
dd if=/dev/sdc of=mbr_dump.img count=1 bs=512
```

This means that it will save the first 512 bytes (which contain the MBR and the partition table) of the hard drive disk of the iPod.

In order to restore them you have to type :
```
dd if=mbr_dump.img of=/dev/sdc
```

3.2.3.2 Firmware

To dump the firmware from the iPod to your computer, you have to type the following command, where you will replace "COUNT" by its value (see part 3.2.2):
```
dd if=/dev/sdc1 of=firmware_dump.img bs=2048 count=COUNT
```

To dump the whole partition, just type:
```
dd if=/dev/sdc1 of=partition_dump.img
```

To install a new firmware on the iPod, you have to type the following command:
`dd if=Firmware-w.x.y.z of=/dev/sdc1`
To put back on the iPod a dump of the whole sdc1 partition, type:
`dd if=partition_dump.img of=/dev/sdc1`

When the firmware is copied on the iPod, you have to unmount the peripheral and unplug it from your computer. Then the decompression and decryption of the firmware will be automatic.

3.2.4 Under Windows

3.2.4.1 A few explanations about the commands

First of all, as *dd* is not a "native" Windows software, you need to download it[1], and to save *dd.exe* into the folder `/Windows/system32/`.

Furthermore, you need to know that the firmware's partition starts at the 64th logical block. Under Linux, there is no problem because the OS recognizes the beginning of the partition. But under Windows there is a problem: in fact the OS considers the iPod as a unique logical partition (ie MBR + two partitions). So if you want to dump a firmware, you have to skip the first 63 blocks (`skip=63`), and if you want to copy a firmware on the iPod, you have to start writing from the 64th block (`seek=63`).

3.2.4.2 Commands

As under Linux, you need to have administrator's rigths to execute these commandes (under Vista / Seven: rigth click, execute as administrator).
Then, in the Windows command prompt, do the following actions:

1. type `dd --list`
2. search in the "NT Block Device Objects" for a sequence like (where size is 2GB, 4GB or 8GB, depending on your Nano version):

```
\\?\Device\(XXXXXXXX)\Partition0
  link to \\?\Device\(XXXXXXXX)\DR23
  Removable media other than floppy. Block size = 2048
  size is 2030043136 bytes
```

3. note the (XXXXXXXX) field (it looks like "Harddisk5" or similar)
4. - to dump the firmware from the iPod type the command, where you will replace "COUNT" by its value (see part 3.2.2):

```
dd if=\\?\Device\(XXXXXXXX)\Partition0 of=firmware_dump.img
   bs=2048 skip=63 count=COUNT
```

   - in the other direction, to install a new firmware on the iPod type the command:

```
dd if=Firmware-w.x.y.z of=\\?\Device\(XXXXXXXX)\Partition0
   bs=2048 seek=63
```

   - the MBR is in the first 512 bytes (before the first partition); to dump it, just type:

```
dd if=\\?\Device\(XXXXXXXX)\Partition0 of=mbr_dump.img bs=512 count=1
```

[1] http://www.chrysocome.net/dd

3.3 Firmware partition

Remark 1: offsets given below are those from the 19.1.1.3 version of the firmware.
Remark 2: all the values in the firmwares are in little-endian. It means you have to read "from the right to the left": if you see "12 34" in the firmware, just read "0x3412". All the values in this report are real ones ("0x3412"), except on pictures.

3.3.1 Firmware structure

Address	Content
0x0	Volume header
0x800	Zeros, Start of volume space
0x4800	Directory entries
0x4940	Zeros
0x5000	*rsrc.fw* file[1]
0x1006000	*osos.fw* file[1]
0x15FD000	*aupd.fw* file[1]
0x172A000	Zeros

Tab. 3.2: Firmware 19.1.1.3 - Overview

Remark: for the 19.1.1.0 and 19.1.1.1 versions of the firmware, the order of the three ".fw" files was *osos.fw* -> *rsrc.fw* -> *aupd.fw*, but since the 19.1.1.2 version, Apple swapped *osos.fw* and *rsrc.fw*...

3.3.2 Volume header

The header starts with some copyright infos (from 0x0 to 0x95). Then, from 0x96 to 0xFF, each bytes has the value 0x2D.

The following bytes and their signification (according to iPodLinux) are:

Address	Content	Signification
0x100	[hi]	???
0x104	0x4000	location of the directory entries
0x108	0x10C	location of the extended header
0x10c	3	firmware format version (see below)

Tab. 3.3: Volume header overview

The "[hi]" may simply be a visual tag to easily detect the beginnning of the block...

According to iPodLinux, the "3" as value for the *firmware format version* field means that the firmware uses Volume Space-relative addressing (all the address given here are relatives to the beginning of the volume space).

The end of the header (from 0x10B to 0x7FF) is filled with zeros.

3.3.3 Directory entries

Directory entries' list starts at 0x4800. It is a kind of allocation table which has three entries of 40 bytes each (plus two free entries at the end). Picture 3.2 shows the directory entries of the 19.1.1.3 firmware.

Table 3.4 lists and explains the different fields with the values of *osos.fw* (from 0x4828 to 0x484F):

[1] see part 3.4 of this report

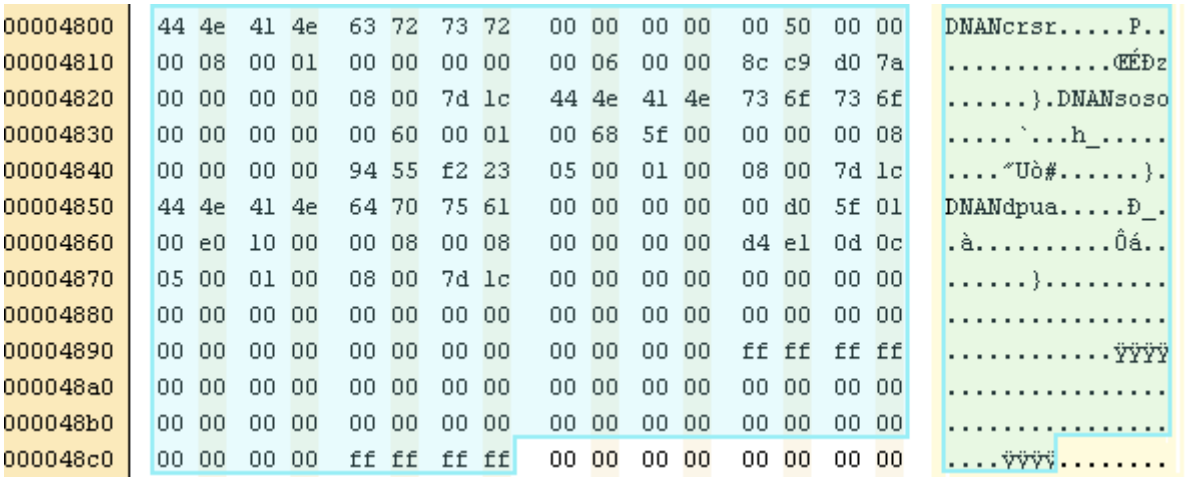

Fig. 3.2: Firmware 19.1.1.3 - Directory entries

Relative address	Value	Signification (according to iPodLinux)
0x0	NAND	type of storage
0x4	osos	file
0x8	0x0	id[1]
0x12	0x1006000	offset where the file begins
0x16	0x5F6800	length of the file
0x20	0x8000000	address where the file is loaded in memory
0x24	0x0	execution start within file
0x28	0x23F25594	checksum of the file
0x32	0x10005	file version
0x36	0x1C7D0008	load address for the file

Tab. 3.4: Firmware 19.1.1.3 - Directory entries for *osos.fw*

3.4 The files osos.fw, aupd.fw and rsrc.fw

3.4.1 Extraction

There are three files in the firmware:

- *aupd.fw*: the Apple NOR flash updater image (encrypted)
- *osos.fw*: the Apple OS (encrypted)
- *rsrc.fw*: the file system (unencrypted)

To extract them we use a tool named *extract2g*, created by Jean-Damien 'JD' Brossillon.
Just follow the instructions below.

3.4.1.1 Under Linux

1. Download the last versions of extract2g.c, extract2g.h and the Makefile from Linux4Nano's SVN[2]
2. Type `make` to compile
3. Type `extract2g -A path_of_the_firmware` to launch extraction

[1] see part 3.4.3 of this report for an explanation of this field
[2] http://svn.gna.org/viewcvs/linux4nano/trunk/tools/extract2g/

3.4.1.2 Under Windows

1. Download the last version of extract2g.exe from Linux4Nano's SVN[1]
2. Type `extract2g -A path_of_the_firmware` (in the Windows command prompt) to launch extraction

Remark: with the extract2g tool, you can specify options for the extraction, type extract2g –help to list them.

3.4.2 Structure and differences between successive versions

3.4.2.1 A few words about rsrc.fw

Table 3.5 shows the structure of *rsrc.fw*. This file is exactly the same in each version of the firmware.

Address	Content
0x0	Header (Full of 0xFF)
0x800	Zeros
0xE00	Beginning of file data
0x1000FFF	End of file data

Tab. 3.5: *rsrc.fw* overview

As *rsrc.fw* is the firmware's ressource file system, it is definitely uninteresting for this project and we won't talk about it anymore.

3.4.2.2 Structure of osos.fw and aupd.fw

The *aupd.fw* and *osos.fw* files have the same structure, which is shown in table 3.6

Address	Content
0x0	Header
0x200	Zeros
0x800	Data Part ("Payload")

Tab. 3.6: *aupd.fw* and *osos.fw* overview

3.4.2.3 Header structure

The next two pictures show the *osos.fw* (picture 3.3) and *aupd.fw* (picture 3.4) headers (from the 19.1.1.0 version of the firmware).
Only the blue and the two pink fields change from a version to an other:

- The blue one is simply the file's length.
- The first pink one is a hash of the decrypted payload
- The second pink one is a hash of the header

The two hash fields *seem* to be HMAC-SHA1 hashes.

[1] http://svn.gna.org/viewcvs/linux4nano/trunk/tools/extract2g/

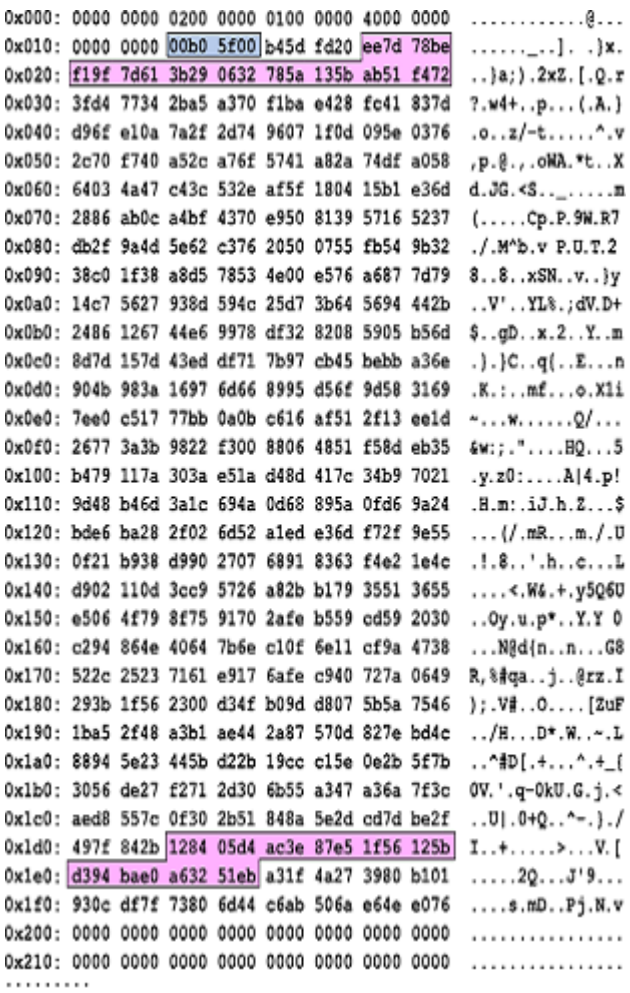

Fig. 3.3: *osos.fw* header

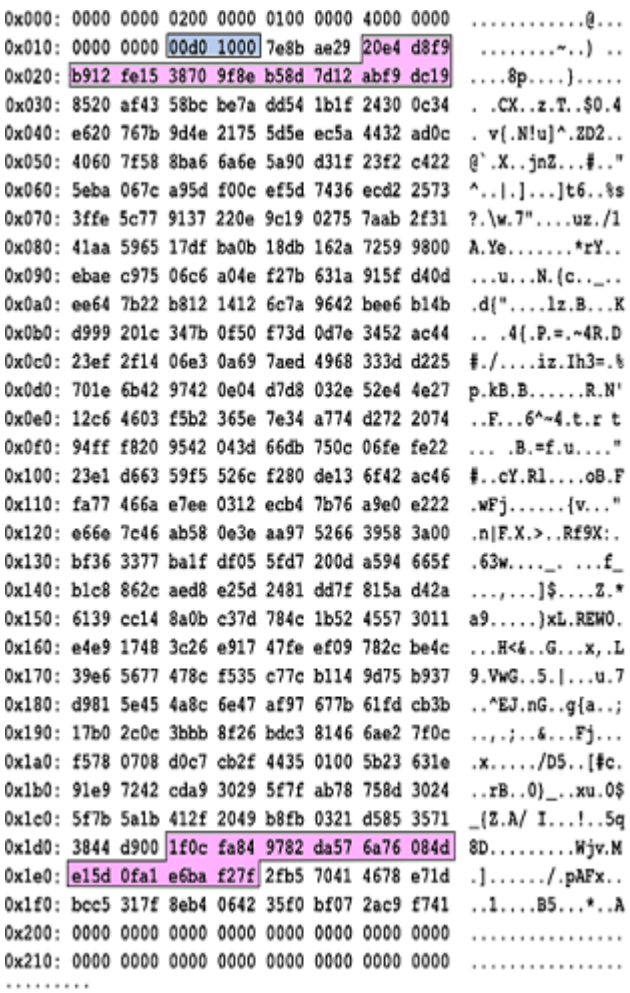

Fig. 3.4: *aupd.fw* header

3.4.2.4 Data structure

There is no header-like structure in the data part (or we did not find it), but the comparison between different versions shows that some of the bytes at the beginning of the data part are similar.
Concerning *osos.fw*, apart from 19.1.1.0 version which has no byte in common with others, the first 1680 bytes (from 0x800 to 0xE8F) are identical each time.
Table 3.7 shows the number of identical bytes at the beginning of the data part for successive versions of *aupd.fw*.

Version	19.1.1.0	19.1.1.1	19.1.1.2	19.1.1.3
19.1.1.0	-	240	32	32
19.1.1.1	240	-	32	32
19.1.1.2	32	32	-	240
19.1.1.3	32	32	240	-

Tab. 3.7: Number of common bytes in different versions of *aupd.fw*

3.4.3 Modifying the content of the firmware

If we modify *osos.fw*, the IN2G will always refuse to boot, but it seems that it is not the case for *aupd.fw*: it is because of the "*id*" flag in directory entries.

This field is used only for *aupd.fw*.
In a complete new firmware (never used, including osos aupd and rsrc), the *id* of *aupd.fw* is set to zero. After a restart, it performs the Flash ROM update and then changes the *id* to 1 (which means that the

iPod is decrypting *aupd.fw* and then flashing it to the Flash ROM).
That explains why no problem occurs if we modify the content of an "used" *aupd.fw*: the iPod does not check *aupd.fw*, as long as id=1.

4 Encryption

4.1 Encryption on previous iPods

4.1.1 Encryption on iPod 4G, 5G, and 5.5G

The IN2G is not the first iPod where cryptography appears: the first one was the iPod 4G. On this model (and on the 5G and 5.5G), only *aupd.fw* is ciphered. The algorithm used is RC4, with a 32 bits key.
The key is hidden in the "security block", which is the first 512 bytes of the file. This security block tells the bootloader if the file is encrypted or not. If it is, the bootloader delivers the key, in order to decrypt it.
The security block contains eight "flags" which can be enabled or disabled. If any of them is enabled, the file is considered as encrypted, and if they are all disabled, the file is unprotected. These eight "flags" are located at the offsets {0x5, 0x25, 0x6f, 0x69, 0x15, 0x4d, 0x40, 0x34}. To get the actual offset in bytes in the block, you have to multiply by 4.
Once the key is found, a RC4 decrypter gives the deciphered file.

These informations comes from the iPodLinux Wiki[(1)] and from the message of Franco Zavatti (who wrote the Java version of the decrypter) on Linux4Nano's mailing list.

4.1.2 Comparison with the IN2G

Unfortunatly, the security block in the IN2G is not the same as the previous one (it is 2048 bytes long). As if it wasn't enough, *osos.fw* and the bootloader are encrypted too. So the method described above cannot be applied.
Another attempt which has been performed is based on noticing that in *rsrc.fw* (which is not ciphered), the first 2048 bytes are filled with 0xFF. So a trial was to change the first 2048 bytes of *osos.fw*, but it hasn't been successful.
After this trial, a more detailed study has been launched.

4.2 Testing if it is a "basic" encryption

Remark: this section is to a large extent a copy from the report made in 2007

4.2.1 Entropy

The entropy is a measure of disorder and correlation between data. The entropy ϵ is computed using the formula $\epsilon = -\sum_{k=0}^{255} \frac{c_k}{n} log_2(\frac{c_k}{n})$ where k corresponds to a character, c_k corresponds to the number of character k occurences, and $n = \sum_{k=0}^{255} c_k$.
A high entropy (here, 8 is the maximum) means that the file looks like a random file, and here, according to the results shown in table 4.1, it leads to this conclusion.

[1] http://ipodlinux.org/wiki/Flash_Decryption (on this page, you can download the AUPD decrypter, wrote for iPod 4G, 5G and 5.5G)

Firmware	Value
osos 19.1.1.0	7.999958
aupd 19.1.1.0	7.999459
osos 19.1.1.1	7.999959
aupd 19.1.1.1	7.999497
osos 19.1.1.2	7.999964
aupd 19.1.1.2	7.999496

Tab. 4.1: entropy values of *osos.fw* and *aupd.fw*

These values shows another thing: an entropy which is that high means that the file is encrypted.

4.2.2 Monoalphabetic substitution

The first algorithm considered is monoalphabetic substitution. Each character (ASCII) corresponds to an other. For instance, adding 13 modulo 256 is a possible encryption function. Any substitution on the ASCII characters is possible too.
This type of algorithm keeps one of the original text's property: the frequency of each character is the "signature" of a text.
A simple program was made to analyze dumped files. The result was that each character appears quite as many time as the others in the *osos.fw* file: the encryption algorithm is more complex than this simple monoalphabetic substitution.

4.2.3 Polyalphabetic substitution

The first assertion has been invalidated. Using several different substitutions could basically give a balanced repartition. Thus, this hypothesis can be assumed. It can be supposed the substitution number is a multiple of 4 (which represents ARM9 instruction length; see part 6.2 of this report).
To assert this hypothesis, the previous program was modified in order to analyze the first bits of each "instruction", corresponding to the opcode.
Again, according to figure 4.1, frequency seems to be the same for each ASCII character. Furthermore, comparing with an analysis (see figure 4.2) done on an ARM binary version of *ls software*, frequencies are really different. Thus, polyalphabetic substitution hypothesis has also been invalidated.

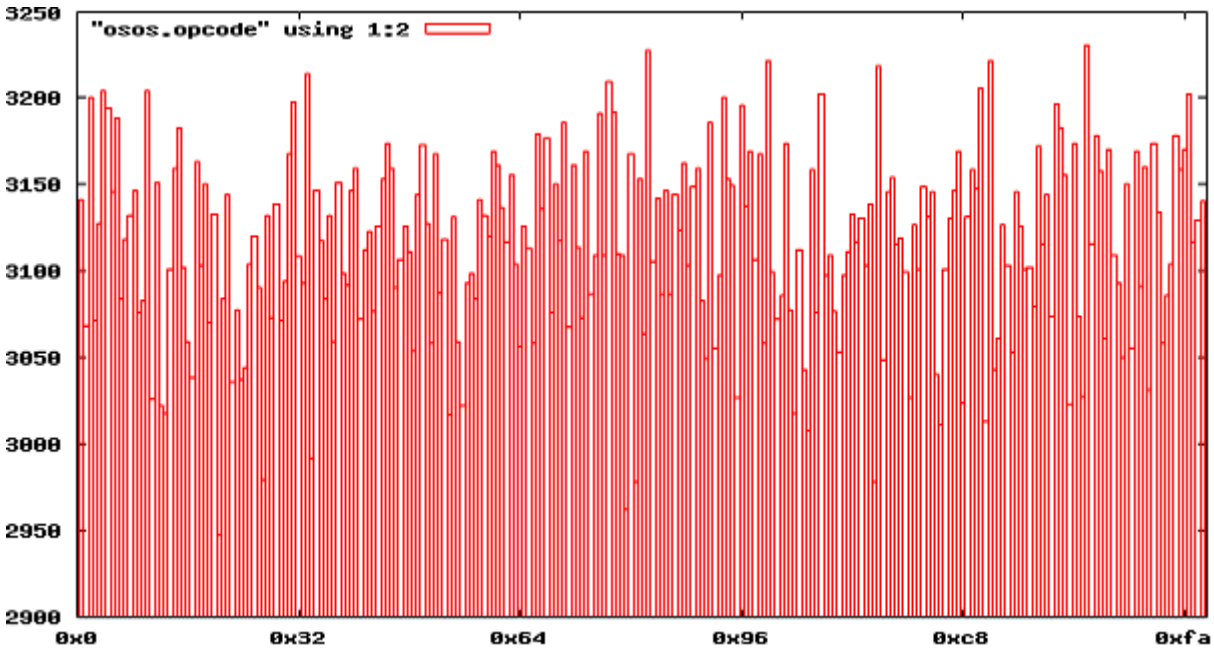

Fig. 4.1: Statistics on *osos.fw* opcodes

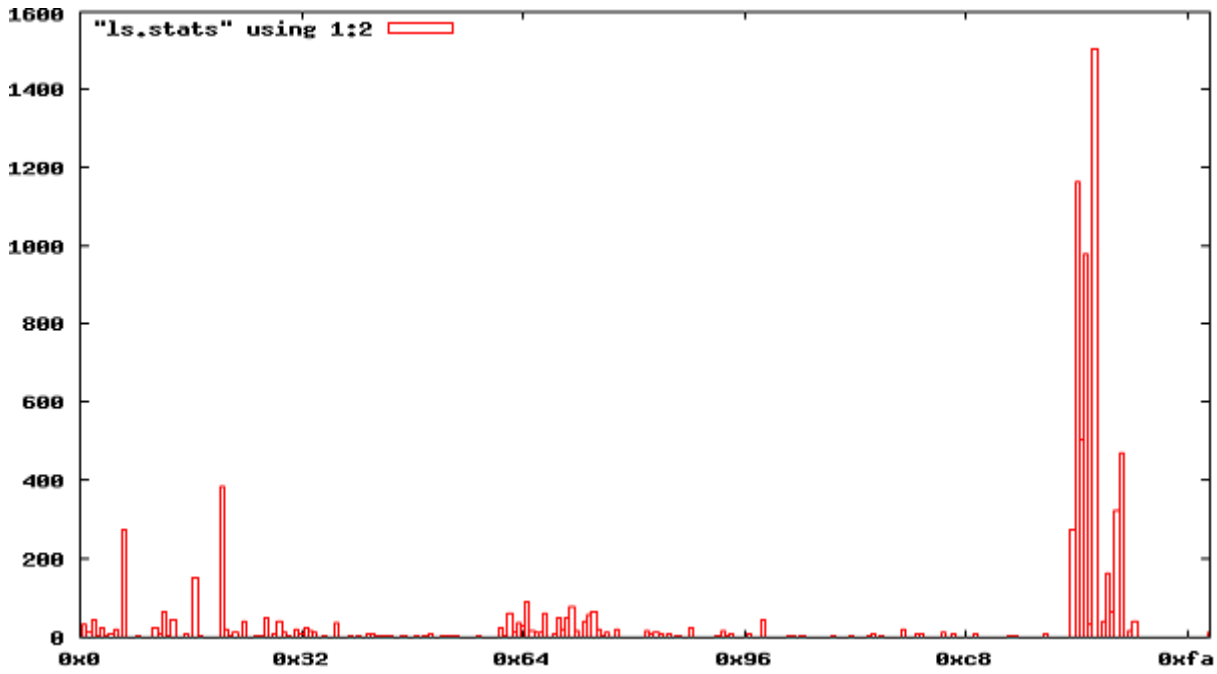

Fig. 4.2: Statistics on ARM opcode (considering ls ARM binary)

4.2.4 Substitution by 4 bytes blocks

An other hypothesis which can be made is the following: files are encrypted using substitution by blocks of a multiple of 4 bytes. A program which reads files 4 bytes by 4 bytes was designed, but no repetitions have been found: this assertion is false too.

4.2.5 Pattern research

It can be supposed that the length of the blocks isn't a multiple of 4. However, finding repetitions of any length in a text requires a lot of memory.
fv program[1] does repetition statistics using hash technics. One more time, this test returns that the file looks like a pure random text...

4.2.6 Compression

An other hypothesis was that the encryption algorithm is a compression algorithm. Some facts reject this idea:

- firmware size does not seem to have been reduced
- ciphered file entropy is too high to correspond to a compression algorithm

Nevertheless, a case could have been possible: a compression plus an encryption. But we now know that it is just encryption without compression.

4.3 Determination of the kind of algorithm used

First of all, we made two hypothesis which are that Apple used the same algorithm and the same key(s) to encrypt each version of the firmware.
These hypothesis are realistic: for different versions of *osos.fw* and *aupd.fw*, there are many similarities. An other thing: we may suppose that there is no reason for Apple to use others, as the coding has not been broken.

[1] http://www.fantascienza.net/leonardo/ar/string_repetition_statistics/string_repetition_statistics.html

As we saw, the firmware is not ciphered with a "basic" encryption. We have now to determine which algorithm is used among all the strongest ones.
There are two categories of algorithms: symmetric-key encryption (private key) and asymmetric-key encryption (public key). As there is no public key here, we focus on symmetric ciphers.

Among symmetric ciphers, there are two sub-categories: block cipher (for example AES) and stream cipher (for example RC4). Finding which kind of algorithm we are facing can be very useful to determine how the firmware is deciphered by the iPod, where the key(s) is(are) located,...
The functionning of block cipher and stream cipher algorithms is different and has a consequence we can use it to help this research: if you change a single byte in a file and apply a block cipher algorithm, only a few bytes near this one in the ciphered file will be modified, but with a stream cipher algorithm, all the following bytes will change.

"Unfortunately", there are some schemes for the block cipher algorithms which do not follow this "rule", like the Cipher Block Chaining (CBC), Cipher FeedBack (CFB),...

According to figure 4.3, the hypothesis of a "basic" block cipher algorithm is wrong.

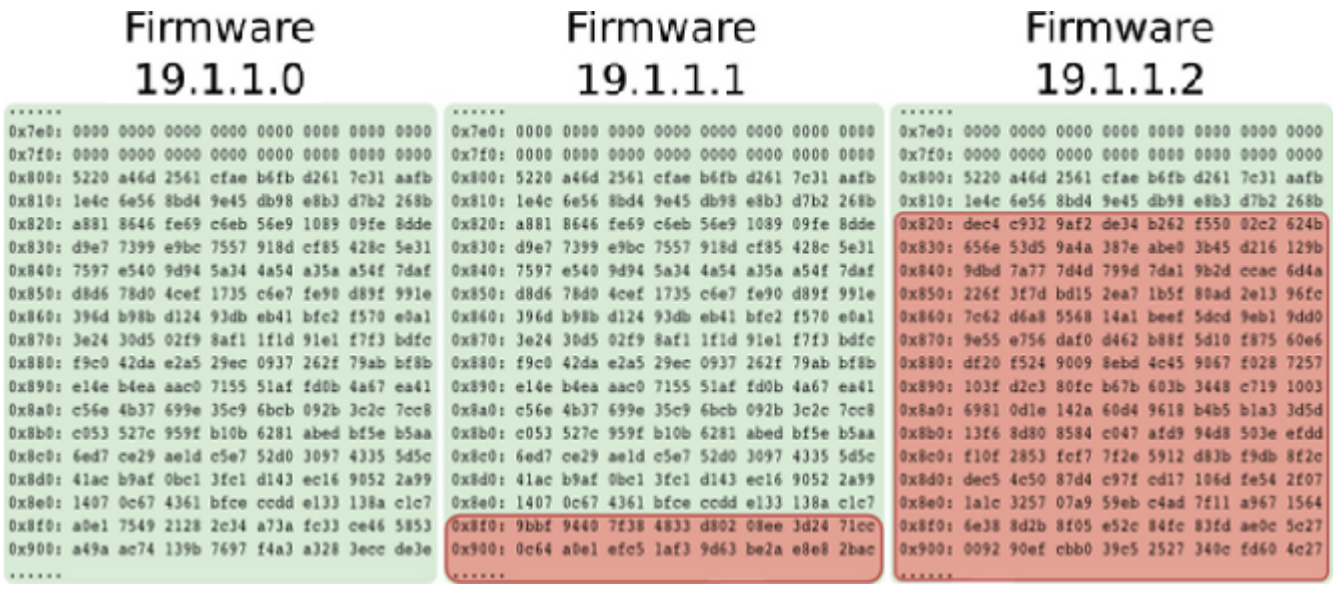

Fig. 4.3: Differences between some versions of *aupd.fw* (in red)

4.4 Current assumptions

4.4.1 Encryption algorithm

At the present time, the assumption is that the firmware is encrypted by AES-CBC$^{(1)}$, with 128 bits long blocks (and a 128 bits long key).
Some reasons point to this conclusion:

- the iPhone is encrypted by *FileVault* (which protects Mac OS X), based on AES-128
- it takes around ten second to decrypt the firmware, which is compatible with the speed of the SOC and the number of operations needed to decrypt a file ciphered by AES-128
- each version of the firmware has a size (in bytes) divisible by 16 (table 3.1 shows these sizes)
- the number of identical bytes at the beggining of the data part of different versions of *osos.fw* is 1680, and for *aupd.fw*, it is a multiple of 16 (see table 3.7)

As the number of common bytes between the versions 19.1.1.1 and 19.1.1.2 of *aupd.fw* is 32, we could think that blocks size is 32, but neither 1680 nor 240 is divisible by 32.
According to the previous facts, blocks length could be less than 128 bits, but it is not very credible because AES runs with 128, 192 or 256 bits long blocks, and choosing a smaller size would give very small advantages (speed,etc.), but would reduce a lot the security.

4.4.2 Keys and Initialization Vector location

The CBC scheme needs two things: a key, of course, and an Initialization Vector (IV). This IV has the same length as the blocks. Some references always set it to 0x0000000000000000, but in facts it can have any 128-bits value.
So, we are here trying to find two 128 bits long integers.

In a previous version (iPod 5G, where only *aupd.fw* was ciphered), a part of the key was hidden in its header. The key was computed by an algorithm located in the bootloader, using the header and a constant variable located in the bootloader.

Nevertheless, things have changed with the IN2G: the key is now located in an "AES unit" (see part 8.3). It allows us to encrypt / decrypt, but we have no access to its value.

4.5 Conclusion

If all the hypothesis above are right, we cannot break the encryption: the only known method, which works in every case, to attack AES is the *brute-force attack*, and it would take years...
But, as we found how to decrypt / encrypt without knowing the key, it is not very important anymore!

[1] http://www.faqs.org/rfcs/rfc3602.html

5 "Notes" buffer overflow

This part is mainly a copy of Linux4Nano wiki[1].

Remark: the leak which was exploited by the buffer overflow is fixed by Apple since the last Nano 4G firmware, and the first one for Nano 5G.

5.1 Noticing

5.1.1 Buffer overflow noticing

On February, 14th 2009, Cory 'cmwslw' Walker first found a bug in the "Notes" folder's feature[2]: when you copy an HTML file with an hyperlink (which respects some conditions[3]), the iPod crashes.
This bug presumably happens because the filesystem of the iPod Nano cannot handle files which contain a very long character string.

5.1.2 Creation of a working overflow file

To reproduce this buffer overflow, you have to create an HTML file with only the following code:
`<a href="STRING">Crash!</a>`
Then, just copy it in the "Notes" folder.
To remove it, you have to put the iPod into disk mod (see appendix A).

The content of the STRING field has to follow some rules:

- it has to be more than 256 chars long (Apple documents this limit, but they do not say it may cause a buffer overflow)
- it cannot be more than 4096 chars long, or the iPod will truncate it (and won't recognize it as a correct link: tag will be missing)
- the file size is limited to ~4kB (3788 bytes for the payload to be exact)
- 64kB files are loaded just after the boot of the nano, however they are not kept in RAM

There are many copies of the buffer, but they can be divided into 2 "types". One being the RAW file data, and one being corrupted by some UTF16 chars.

The HTML file is converted to UTF-16 first. This limits the possible char sequences. The best thing to have most charset possibilities is to encode the exploit directly to UTF16.

A consequence of this UTF-16 conversion is that there are some forbidden values:

- 0xFEFF: UTF16 BOM
- 0xD800 up to 0xDFFF
- 0x0000: would stop string processing

[1] http://l4n.clustur.com/index.php/Getting_execution
[2] it is basically a HTML browser included in the iPod
[3] These conditions were found by fuzzing with files of different contents and sizes

5.1.3 Link overflow

After loading the file, the links are then checked against the file system. Many modified copies of this string are present on the stack. We could identify the most important steps of this process, until the string overflows in the stack (order could be a little different):

- First, the link is extracted from the file, and copied to some heap or fixed buffers
- The link is converted to UTF8. Every char over 0x7F is encoded in many bytes, making some useless code
- Then it is passes through an uppercase function
- The URL encoding is decoded: %xx values are converted to their equivalent (limited to valid UTF8 or alike)
- Finally, this link is copied in a limited buffer which is located on the stack. By putting a return address repetitively in the link, the processor will jump to this address.

For convenience, the return address is always encoded using %xx URL encodings. This avoids problems with some special chars and with lowercase chars. Possible values are $00 < xx \leq 0x7F$ (the unescaped chars seem to be transcoded from ISO-8859-1 to UTF8 again).

5.2 Exploiting, getting execution

To exploit, we used JTAG (see part 7) to determine the correct paddings and return addresses of the buffers.

At http://f4eru.free.fr/8701/Notes_overflow_example.zip, you can download an example of a working overflow file. The file *Brokenlink.htm* contains first an UTF16 BOM, then "AA" as padding, then the overflowing link (return address is 08640D60), then a NOP (opcode E1A01001) landing zone, and finally a "while(1);"
This while(1) does not freeze or reset the iPod, but instead just locks up the background task. You can still scroll menus, but the iPod will freeze as soon as you press "play" or if you enter the "Notes" menu, etc.
The processor arrives to the Notes in supervisor state, with interruptions activated (menu scrolling), etc.

Remark: Direct Memory Access could cause cache problems, but it does not with the call of some cache coherency CP15 functions.

5.2.1 Dumping memories

For dumping, first the cache was used (JTAG dumps), but very soon it turned out that the UART was more flexible. All these dumps can not be published here, due to copyright issues.

5.2.2 UART

The UART is *the same* than described in the datasheet. In fact, we have noticed one difference: the clock divider registry has only 8 bits instead of 16.

See http://pargon.nl/?p=6 how to build an UART cable.

Christophe 'tof' Riehl's complete setup is a little bit more complex (see photo 5.1[(1)]):

[1] full sized photo is available at http://l4n.clustur.com/images/6/67/Nanofighter.jpg

- left board : DLC5 JTAG interface, modified for reset and USB switching
- right board : some programmer board, only the ST232 is used
- upper board : this was the JTAG scanner, now only the power supply and 5V regulator are used
- middle board : all the switching stuff

Fig. 5.1: "nanofighter"

To automatically enter DFU mode, he wired transistors to the USB 5V line, and to the "play" and "enter" buttons of the clickwheel.

5.2.3 USB

Because UART needs HW, USB will be used to debug in the future.

Part III

Hardware study

Table of Contents

6 Components **29**

6.1 Components list . . . 29
6.1.1 The System On Chip . . . 29
6.1.2 NAND Flash Chip . . . 29
6.1.3 Utility Flash Memory . . . 29
6.1.4 Mobile SDRAM . . . 29
6.1.5 Audio Codec . . . 29
6.1.6 Voltage Regulator . . . 29
6.1.7 USB Power Manager . . . 29
6.1.8 Pictures . . . 30
6.2 ARM 940T Survey . . . 31
6.2.1 The SOC's internals . . . 31
6.2.2 About the instruction set . . . 32

7 JTAG research **33**

7.1 JTAG definition, and its usage . . . 33
7.1.1 A short definition . . . 33
7.1.2 Using JTAG for testing . . . 33
7.2 JTAG location . . . 33
7.3 JTAG cache dumps . . . 34
7.4 Conclusion . . . 34

6 Components

Remark: this chapter is an update of the 2007 report.

6.1 Components list

At the beginning of the project we used to only consider data from other IN2G studies, which was far from perfect. Based on autopsy facts, this is an update concerning what we know about the IN2G.

6.1.1 The System On Chip

The SOC is part numbered "337S3291 8701 ARM", and it is Apple marked on top (see figure 6.3). According to EETimes[1], part "337S3291 8701" corresponds to a Samsung Electronics Co. Ltd. S5L8701B05 ARM processor[2]. In fact it appears that this SOC contains an ARM940T processor (see part 6.2 of this report), 50KB boot ROM, 176KB SRAM, external RAM, flash and LCD controllers, USB (1.1-host; 2.0-function) and some other parts.

6.1.2 NAND Flash Chip

This chip is used to store program data (audio, photos, games). It is not the same in every IN2G, partly because of the different available capacities.
The one we found here is part numbered "HY27UW08BGFM", from Hynix Semiconductor (see figure 6.4). It is a 4GB MLC NAND flash memory. Several versions have been used in iPods.

6.1.3 Utility Flash Memory

The utility flash is part numbered "SST39WF800A", and is made by Silicon Storage Tech (see figure 6.5). It is a 8 Mbit multi-purpose flash[4], which stores the bootloader, Disk mode, Diagnostic mode, DFU mode and the code to flash it. tof has managed to extract this data (and it led to iLoader: see part 8)

6.1.4 Mobile SDRAM

The external Samsung SDRAM is a 256Mbits flash chip, and part numbered "K4M56163PG"[4] (see figure 6.6). This is the same chip used in the iPod Nano 1G.

6.1.5 Audio Codec

This chip is part numbered "Apple 338S0310 68CXST8" (see figure 6.7). According to SeekingAlpha[3] it used to be an audio codec formerly supplied by Wolfson Micro, probably a Wolfson WM8750L[4].

6.1.6 Voltage Regulator

This LM34910[4] step down switching regulator manages the power supply. This is a low cost buck bias regulator capable of supplying 1.25A to the IN2G. Figure 6.8 shows the chip, pinout is easily recognisable.

6.1.7 USB Power Manager

This chip part numbered "LT 6I 4066" or "LTC4066"[4] is from Linear Technology (see figure 6.9). It is designed to manage Li-Ion battery charge through USB, which requires very accurate voltages. It also

provides USB power management and short circuit protection.

6.1.8 Pictures (taken during the "autopsy" of the IN2G)

6.1.8.1 Overview

Pictures 6.1 and 6.2 show the IN2G once opened (first one is the front view, and second one the back):

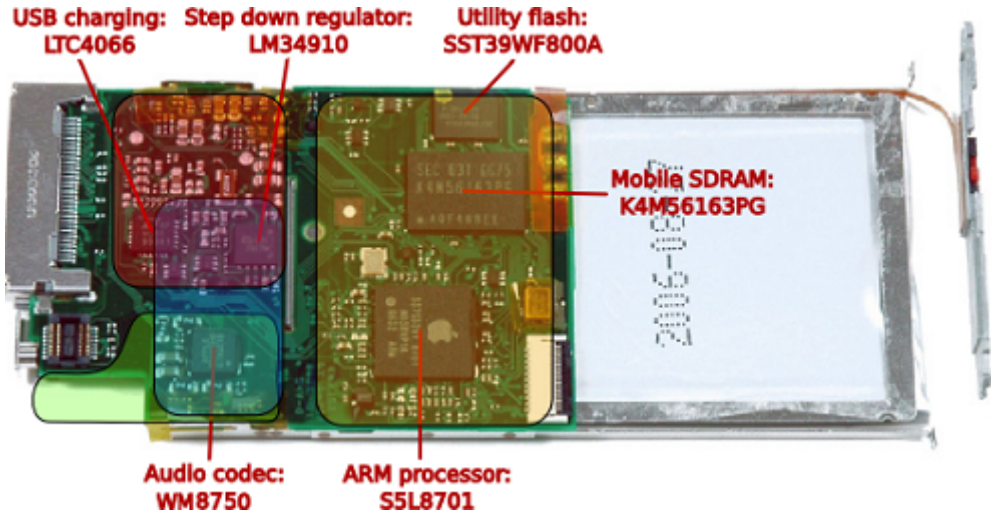

Fig. 6.1: Front side of the IN2G

Fig. 6.2: Back side of the IN2G

6.1.8.2 Details

Next figures show each chip separately:

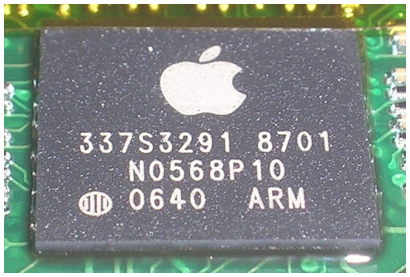

Fig. 6.3: The System On Chip

Fig. 6.4: The NAND flash chip

[1] http://www.eetimes.com/news/latest/showArticle.jhtml?articleID=193000601
[2] http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=212&partnum=S5L8700
[3] http://seekingalpha.com/article/16904-component-makers-who-won-placement-in-the-new-ipod-nano
[4] you can download its datasheet from Linux4Nano website

Fig. 6.5: The utility flash memory

Fig. 6.6: The external SDRAM

Fig. 6.7: The audio codec

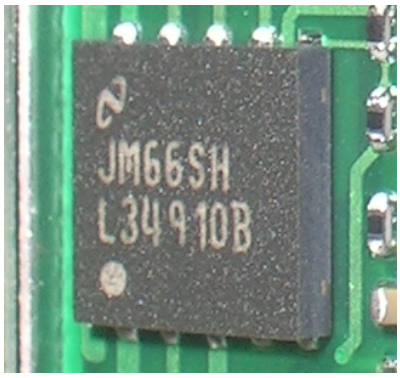

Fig. 6.8: The voltage regulator

Fig. 6.9: The USB power manager

6.2 ARM 940T Survey

We are now focusing on the IN2G main chip.

6.2.1 The SOC's internals

The iPod main chip is the Samsung S5L8701B5 ARM SOC (System On Chip), which will be refered as SoC.
This SoC contains (among many other things):

- an ARM940T[(1)] application processor clocked at 200 MHz with 256KB of SDRAM
- a flash chip controller

The flash chip controller does not interest us so we will focus on the ARM940T application CPU. It is using the ARM-9 instruction set.

[1] visit http://infocenter.arm.com/help/topic/com.arm.doc.ddi0092b/DDI0092B_ARM940T_TRM.pdf to get a technical reference manual

6.2.2 About the instruction set

The ARM instruction set is really a 'nice' one because it is well built.
The best example, and the most useful in this case, is the first 4 bits of an instruction (which looks like "CCCCXXXX XXXXXXXX XXXXXXXX XXXXXXXX"), as shown in table 6.1

Instruction Bitmap	No	Condition Code	Status Register
0000xxxx xxxxxxxx xxxxxxxx xxxxxxxx	0	EQ(Equal)	Z
0001xxxx xxxxxxxx xxxxxxxx xxxxxxxx	1	NE(Not Equal)	Z
0010xxxx xxxxxxxx xxxxxxxx xxxxxxxx	2	CS(Carry Set)	C
0011xxxx xxxxxxxx xxxxxxxx xxxxxxxx	3	CC(Carry Clear)	C
0100xxxx xxxxxxxx xxxxxxxx xxxxxxxx	4	MI(MInus)	N
0101xxxx xxxxxxxx xxxxxxxx xxxxxxxx	5	PL(PLus)	N
0110xxxx xxxxxxxx xxxxxxxx xxxxxxxx	6	VS(oVerflow Set)	V
0111xxxx xxxxxxxx xxxxxxxx xxxxxxxx	7	VC(oVerflow Clear)	V
1000xxxx xxxxxxxx xxxxxxxx xxxxxxxx	8	HI(HIgher)	C and Z
1001xxxx xxxxxxxx xxxxxxxx xxxxxxxx	9	LS(Lower or Same)	C and Z
1010xxxx xxxxxxxx xxxxxxxx xxxxxxxx	A	GE(Greater or equal)	N = V
1011xxxx xxxxxxxx xxxxxxxx xxxxxxxx	B	LT(Less Than)	N = V
1100xxxx xxxxxxxx xxxxxxxx xxxxxxxx	C	GT(Greater Than)	(N = V) and Z
1101xxxx xxxxxxxx xxxxxxxx xxxxxxxx	D	LE(Less or equal)	(N = V) or Z
1110xxxx xxxxxxxx xxxxxxxx xxxxxxxx	E	AL(Always)	True
1111xxxx xxxxxxxx xxxxxxxx xxxxxxxx	F	NV(Never)	False

Tab. 6.1: ARM "condition code" list

That 4-bits code, called "Condition Code", allows one to run every single ARM instruction conditionally. In this case that condition code can be very interesting because we are looking for bit patterns in firmware code. So the first 4 bits of an instruction may have a special distribution: for example we may think that some opcodes are not often used, like NV.
See part 4.2.3 of this report to read the results of this analysis...

7 JTAG research

7.1 JTAG definition, and its usage

7.1.1 A short definition

JTAG ("Joint Test Action Group") is an IEEE standard, which was originally designed for testing printed circuit board assemblies.
But it is now also used for accessing sub-blocks of integrated circuits, making it an essential mechanism for debugging embedded systems.

7.1.2 Using JTAG for testing

7.1.2.1 Boundary scan

The technique of boundary scan is designed to facilitate and automate the testing of electronic digital maps: knowing the electric scheme of the componant, we apply logical signals on the input pins and check if the output corresponds to what it is supposed to be. That way we can detect any problem due, for example, to a bad solder, etc.
To allow this, each I/O pin is not directly connected inside the digital component, but through a "JTAG adapter", allowing to use it independently of its initial function.

7.1.2.2 Applications

Using JTAG, it is also possible to test (at least partially) combinatorial logic functions, even if there are not JTAG-compatible chips in them.
Another possibility is to test memories by writing and then reading these test values.

And now it becomes interesting for reverse engineering: JTAG can be used to write data into flash memory, and to dump its content.
As a buffer overflow was discovered (see part 5), finding the JTAG location could allow us to follow this overflow in memory, to find the return address, and then to exploit it.

7.2 JTAG location

The JTAG was found after searching with a JTAG bruteforce scanner (wrote by tof). There were a lot of problems, including the scanner not working properly, and a nTRST pin.
But now we have the locations of the pins: they are basically available on the DOCK connector after putting in place some jumpers (2 for nTRST, 1 for other pins).

After connecting a xilinx parallel cable, and installing OpenOCD[1], we can try to connect to the JTAG: **The screen freezes directly when we use the JTAG**. This seems to be a protection against hackers, but it could also be an issue with OpenOCD (even if we highly doubt of this hypothesis). In fact, the ARM 940T processor is still fully functional, but it gets disconnected from the main bus, all memories are not reachable any more. The only memory preserved are the data and instruction caches.

The fullsized versions of the pictures showing the JTAG location are available on the Linux4Nano wiki:

- top image at `http://l4n.clustur.com/images/1/19/Top_annote.jpg`
- bottom image at `http://l4n.clustur.com/images/f/fe/Bot_annote.jpg`

[1] OpenOCD user's guide is available at `http://openocd.berlios.de/doc/pdf/openocd.pdf`

7.3 JTAG cache dumps

As the caches are mainly alive, we focused first on dumping whatever they contained (Dcache only can be dumped, the Icache can only give the indexes).
We used some OpenOCD and bash scripts[1]. The command "dc" dumps the Dcache, "ic" shows the icache indexes (be careful, these values can be corrupted due to the memory bus disconnection).
We used statistics on many dumps to have helpful dumps.

Please note that the DLC5 cable was modified to include a nSRST pin, and openOCD was recompiled for this. It is a desirable feature to have a reset. nTrst was simply tied to the 3.0V power supply, it is just not necessary. Also, one important thing is to cut the power supply during reset, with a MOSFET, for example. If this is not done, the iPod can often go to a "broken battery" state, where the processor thinks the successive resets are due to a defective battery.

7.4 Conclusion

With the knowledge of the JTAG location, tof did Dcache dumps using the exploit crashing ("Notes" overflow: see part 5), to figure out how it works.
Finally he managed to get some code running that way, and used it to copy bootrom contents to the data cache, and then read out the data cache via JTAG. Its analysis led to the creation of a software: *iLoader* (see part IV).

[1] download them at `http://f4eru.free.fr/8701/openocd_config.zip`

Part IV

iLoader and RockBox firmware

Table of Contents

8 Towards iLoader **37**
- 8.1 Introduction . . . 37
- 8.2 The "Notes" overflow exploit . . . 37
- 8.3 The encryption / decryption process . . . 37
- 8.4 iBugger . . . 38

9 Versions **39**
- 9.1 NAND-based iLoader . . . 39
  - 9.1.1 History . . . 39
  - 9.1.2 Processing . . . 39
- 9.2 NOR-based iLoader . . . 39
  - 9.2.1 History . . . 39
  - 9.2.2 Processing . . . 40

10 Tutorial about iLoader **41**
- 10.1 iLoader Installation . . . 41
  - 10.1.1 Under Linux . . . 41
  - 10.1.2 Under Windows . . . 42
- 10.2 Installing RockBox . . . 42
- 10.3 Making Apple firmware work . . . 43
- 10.4 Updates . . . 43
- 10.5 Uninstallation . . . 43
- 10.6 Known issues . . . 43

8 Facts and tools which led to iLoader

8.1 Introduction

Booting code through the "Notes" exploit has proven to be too uncomfortable in the long term, as you break the Apple firmware that way, but still have its non-neglegible bootup times. The RockBox bootloader is faster, but still too slow.
This is why iLoader has been developed. iLoader replaces the whole firmware starting from the second level bootloader, and thus gets booted up directly by the bootROM. It then shows a boot menu and allows you to boot different firmware images, which can be stored on the data partition to allow easy updates. The boot menu of iLoader is fully configurable.

8.2 The "Notes" overflow exploit

After the observation of the "Notes" buffer overflow (see part 5), and finding the JTAG location (see part 7), the objective was to make a dump of the bootROM (and the NORboot of course)[1].
Unfortunately, accessing the memory was not possible that way, so tof did Dcache dumps to figure out how it works.
He finally managed to run some code, and used it to copy bootROM contents to the data cache, and then read out the data cache via JTAG.
Then, cmwslw wrote a tool, named *dumpsorter.py*, to collect the dumped data and make a complete bootrom dump from it.

Once that was done, tof and Michael 'TheSeven' Sparmann worked on getting the UART up and running to be able to dump more things.
In parallel, TheSeven started disassembling the bootROM, and writing a usb driver for that thing.
After series of trials and errors, that was also running, and TheSeven had a first version of *iBugger* (see part 8.4), which could barely do anything.
He developed most of that on the *nanofighter* (see part 5.2.2), as that one had a UART- and JTAG-enabled iPod connected to it, and at first the only feedback we had was whether the iPod crashed or freezed.

8.3 The encryption / decryption process

Without the knowledge of how to encrypt/decrypt, all this would have been useless.
But, while TheSeven was disassembling the bootROM, he found that there is an hardware AES and (probably) HMAC-SHA1 unit, which allows us to encrypt/decrypt a bootloader for example, even if we do not know the key!
It is possible for one reason: the kind of cryptography used in the IN2G is symmetric cryptography. It means that the same key is used to encrypt and decrypt (it is not a public key system). If it was an asymmetric cryptography, there would be a private key which would not be in the IN2G.
He even noticed that there is a global ("GID") key and a unique (per device, "UID") key.
So, using this crypto engine, TheSeven just encrypted[1] iLoader on his own iPod and shared the encrypted binary, which is included in the installer file (notice that RockBox is not encrypted, as the new bootloader supports unencrypted payloads).

[1] the bootROM is the ROM inside the processor, that loads NORboot, and the NORboot is the Apple bootloader (in the NOR flash)

8.4 iBugger

Basically iBugger is a tool which permits to upload/download binaries between the iPod memory and a PC (that is also the first function that was implemented).
The next step in the evolution of it was the LCD driver : you could tell it to copy some graphics from memory to the screen. Then came I2C[(2)], which finally allowed to access the power management unit and thus control the backlight and other things.

In fact while iBugger loader is just polling the USB controller in an endless loop, iBugger core is interrupt-driven and thus can operate in the background, while other code is running. In that way, you can debug your code by halting/resuming it, dumping the current state of it, modifying it on the fly etc. iBugger core also implements a debugging console that can be written to using a *syscall* that is basically doing an equivalent to `printf` to it. The data will be buffered and transferred via USB to a console application on the PC. The console is bi-directional, but we can notice that only the "iPod to PC" part has been implemented on the PC side, because TheSeven did not ever need to use the other direction yet. iBugger is coded in ASM as the loader needed to fit into a "Notes" exploit payload, which only allows approximatively 4KB of space.

[1]Scripts used are ipodcrypt.py, libipodcrypto.py which are findable on http://l4n.clustur.com/
[2]I2C is an interconnect protocol between chips; you can refer to Wikipedia for more details

9 Versions

9.1 NAND-based iLoader

9.1.1 History

After the release of iBugger, we could dump some debug information through the UART while trying to get USB working using the first iBugger version. TheSeven then explored further pieces of the hardware. The next step was the LCD.
After that, he figured out how to control the backlight. More and more drivers were written, and the RockBox port was started.
Once he had a working flash driver, he began hacking on a read-only FTL. After reading disassemblies, and coding/debugging, he got that mostly working.
Then he started to develop a first version of iLoader, that was booting from NAND (it was still way before Dave 'linuxstb' Chapman wrote the RockBox bootloader for nano 2g).

9.1.2 Processing

During installation, the "osos" file on the firmware partition is renamed to "osbk" (osos backup), and iLoader itself will be written to a newly-created osos file.
When the iPod boots, Apple bootloader will thus load iLoader from the firmware partition, which can then decrypt and run the Apple firmware (from the "osbk" file), or an arbitrary unencrypted third-party firmware.
At that point, iLoader did not touch the NOR flash yet, so it relied on the Apple bootloader in that flash for booting. That way, if anything went wrong, we could just recover using disk mode, as everything we had overwritten can be accessed using the mass storage device.
The disk mode button combination will be checked for by the Apple bootloader before iLoader gets booted, so we didn't need to care about that either.
The disadvantage of that method is of course the way longer boot time, because that Apple bootloader is not really fast.

9.2 NOR-based iLoader

9.2.1 History

After the first realease the FTL was ported to RockBox.
Once again, a whole bunch of drivers was being written, and RockBox was getting more and more usable.
Some day, TheSeven implemented read-write support for the FTL, which took several months of debugging until it worked flawlessly.
Once RockBox was mostly working, TheSeven started researching how to use the DFU modes, and how to flash data to the NOR flash and recover from corruption.
Then he worked out which additional hardware init steps are needed when taking over control earlier, and worked on the code doing them.
Once that part ("NORloader") was completed, he ported over iLoader to NOR, and then added a trivial scripting engine to allow for customization.

9.2.2 Processing

To kick off installation, "osos" will be renamed to "osbk" as with the old iLoader, but this time, some installation code will be written to "osos".
During installation, the "osbk" image from the firmware partition and the "diskflsh" and "diagflsh" images from the NOR flash will be loaded to memory and will be decrypted.
Then "osbk" will be patched (to fix some Apple bugs regarding superfloppy formatting and to remove the EU volume limit) and saved as `/iLoader/appleos.bin`.
Then "diskflsh" and "diagflsh" will be compressed and saved to the new (trivial) file system that will be created on the NOR flash during installation.
The Apple bootloader will be replaced by NORloader, which will be decrypted and started by the processor's bootROM. It will initialize some hardware and execute iLoader (which is not encrypted any more), unless the hold button is switched on quickly to enter the recovery stub (which is basically an iBugger loader, and is used to reduce the bricking risk).
iLoader itself and its "welcome screen" theme will also be installed to the NOR flash.
The firmware partition on the NAND flash won't be needed any more, so it will be removed, and the FAT32 data partition will be resized to span the whole device. This allows to reclaim roughly 100MB of otherwise unused space.
iLoader will then look for a theme file, which will control its further behavior.

10 Tutorial about iLoader

This tutorial was written on April 2010. It may be outdated now.
Please refer to Linux4Nano wiki to check the last version:
http://l4n.clustur.com/index.php/ILoader_Howto

It explains how to install iLoader, RockBox and the Apple firmwares **on iPod Nano 2G *only***. Nevertheless, the drivers developpement is not finished[1]. If you want to help, go to the RockBox SVN[2], and on the IRC channel **#linux4nano-dev** at **FreeNode** (irc.freenode.net).

10.1 iLoader Installation

This might *semi-brick* your device! We have taken all possible measures to prevent it, but if norloader (and its recovery mode) should fail for some reason (which has not happened yet), you *will* have to open your device to recover.

BEWARE: As usual, these instructions are supplied in the hope that it will be useful but WITHOUT ANY WARRANTY! Do not blame us if you crash your data or even brick your iPod. You should also read through the "Known Issues" section below.

10.1.1 Under Linux

Caution: the iLoader installation has to be done on a working and original firmware. If there is any doubt, you should make a recovery from iTunes or see part *3.2* before the installation.

1. Fetch *ipodpatcher*[3] - there are versions for Linux (32-bit and 64-bit), Windows and OS X.
2. Fetch *installer.ipodx*[4].
3. Put those two in the same folder.
4. Type `chmod +x ipodpatcher`
5. Connect your iPod to your computer.
6. Open a terminal (or equivalent) and `cd` to the directory of those files.
7. Type the following command: `./ipodpatcher -a installer.ipodx`
8. Now safely eject your iPod and unplug it. The iPod will reboot and start the installation, which can take up to 5 minutes.
9. Plug your iPod in, press *Menu* and then select *Disk Mode*.
10. When the installation is complete, it will ask you for a theme. You can download one at http://l4n.clustur.com/index.php/ILoader_Themes. Then, you have to unzip it and copy what is in the folder "iLoader" created that way, to the same folder "iLoader" on your iPod.
11. In this "iLoader" folder on the iPod, will be a file named "NORFLASH.BAK". **Copy that file to a safe place! It is your only way back! Apple *does not* offer this as as download, and it is *device-specific*!**

[1] visit http://www.rockbox.org/wiki/IPodNano2GPort#Driver_status to see the drivers developpement status
[2] http://svn.rockbox.org/viewvc.cgi/trunk/firmware/target/arm/s5l8700/ipodnano2g/
[3] http://download.rockbox.org/bootloader/ipod/ipodpatcher/
[4] http://bit.ly/bl7q12

12. Now you can move on to installing RockBox! (keep your iPod plugged in)

Please report your testing results on http://l4n.clustur.com/index.php/ILoader_Testing_Results. Thanks!

10.1.2 Under Windows

Caution: the iLoader installation has to be done on a working and original firmware. If in doubt, you should make a recovery from iTunes or see part *3.2* before the installation.

1. Fetch *ipodpatcher*[1] - there are versions for Linux (32-bit and 64-bit), Windows and OS X.
2. Fetch *installer.ipodx*[2].
3. Put those two in the same folder.
4. Connect your iPod to your computer.
5. Open a command terminal and `cd` to the directory of those files.
6. Type the following command: `ipodpatcher -a installer.ipodx`
7. Now safely eject your iPod and unplug it. The iPod will reboot and start the installation, which can take up to 5 minutes.
8. Plug your iPod in, press *Menu* and then select *Disk Mode*.
9. When the installation is complete, it will ask you for a theme. You can download one at http://l4n.clustur.com/index.php/ILoader_Themes. Then, you have to unzip it and copy what is in the folder "iLoader" created that way, to the same folder "iLoader" on your iPod.
10. In this "iLoader" folder on the iPod, will be a file named "NORFLASH.BAK". **Copy that file to a safe place! It is your only way back! Apple *does not* offer this as as download, and it is *device-specific*!**
11. Now you can move on to installing RockBox! (keep your iPod plugged in)

Please report your testing results on http://l4n.clustur.com/index.php/ILoader_Testing_Results. Thanks!

10.2 Installing RockBox

1. First you need to download RockBox. The latest RockBox build, updated automatically on each source code change, is available[3]. Remember this a a development build and can contain bugs.
2. Then unzip this file to the root of your iPod (be carefull the folder is ".rockbox" so it is hidden on Linux, you must keep it in that way).
3. Eject safely your iPod then unplug it (you might have to reboot your iPod by holding down the Menu and Select buttons after that)
4. Then, from iLoader, select the play/pause button to boot RockBox.

If you want more information concerning RockBox, take a look at its current status[4]

[1] http://download.rockbox.org/bootloader/ipod/ipodpatcher/
[2] http://bit.ly/bl7q12
[3] http://build.rockbox.org/data/rockbox-ipodnano2g.zip
[4] http://www.rockbox.org/wiki/IPodNano2GPort

10.3 Making Apple firmware work

The Apple firmware won't work out of the box because whoever wrote it, their file system code mixed up decimal and hex at some point. In order to fix it, you need to patch 2 bytes in the APPLEOS.BIN file:

- offset 0x64d48: replace 0x2b by 0x43
- offset 0x64d54: replace 0x34 by 0x52

If you also want to remove the EU volume limit ("uncap"):

- offset 0x3acd8: replace 0x01 by 0x00

These offsets are for the most recent firmware verison (1.1.3). The offsets for older versions might differ.

10.4 Updates

The theme and iLoader itself can usually be updated independently.

1. To update the theme, just re-download it and extract it to your iPod, overwriting the old files.
2. To update iLoader, fetch *installer.bin*[1] and *update.cfg*[1].
3. Copy both files to the iLoader directory on your iPod.
4. Reboot to the iLoader menu, press the *Menu* button to get to the advanced menu, and select `Update iLoader`.

10.5 Uninstallation

1. Fetch *uninst.bin*[1] and *uninst.cfg*[1].
2. Copy *uninst.bin* and *NORFLASH.BAK* (if it is not there any more) to the iLoader folder on your iPod.
3. Delete *iloader.cfg*, copy *uninst.cfg* to the iLoader folder and rename it to *iloader.cfg*.
4. Unmount and unplug your iPod and wait few seconds (you might have to reboot your iPod by pressing the *Menu* and *Select* buttons).
5. There should be a new menu, select `run nandfcsk` and wait.
6. You may have to restore your iPod (or to download a working firmware and copy it: see part 3.2)

10.6 Known issues

- **If something should go terribly wrong, this might semi-brick your iPod. Go to L4N IRC channel (#linux4nano-dev at irc.freenode.net) for recovery instructions.**
- If it should complain that it cannot mount the FTL, unplug it, reset it and boot into *Disk Mode*, **do not connect it to the PC**, and just reset it again when it says *OK to disconnect*. It should work now. If not, please let us know.
- ~~There is currently a very high rate of flash problems. We're still investigating what the root cause is. Just hit `run nandfsck` and wait some seconds. If that does not help, run *Disk Mode*.~~ **Fixed**.
- Some iPod accessories (especially Nikepod) may refuse to work if iLoader is installed. (This could also be worked around if need be.)

[1] http://bit.ly/bl7q12

- Do not reboot via the Menu+Select key combination shortly after you have booted up the Apple firmware for the first time after installing iLoader. If you do, it will probably not save its settings and start up with the language selection menu again the next time you boot it. We do not know at which point it will save the settings, but we have found a trick: Just connect the iPod via USB, add or remove a file, and properly unmount and unplug it. This will cause a controlled reboot, which will save the settings.
- Depending on your file system contents, it is possible that all data on the iPod will get erased during the installation.
- iTunes will probably want to restore your iPod, as with every other firmware replacement.

Part V

Appendices

A Modes and keys combinations

Contents

A.1 Description 46
A.1.1 Reboot iPod 46
A.1.2 Disk Mode 46
A.1.3 Diagnostic Mode 46
A.1.4 Device Firmware Upgrade Mode 47
A.2 Combinations 47
A.2.1 1st, 2nd, and 3rd Generations 47
A.2.2 Mini, 4th, Photo, Nano, and 5G Generations 47
A.2.3 Pictures 48

We would like to thank iPodLinux: this is mainly a copy of the wiki "Key Combinations"[1] page.

A.1 Description

A.1.1 Reboot iPod

Forces a power cycle. Use this whenever your iPod freezes, or if you want to switch between third-part firmware and Apple one.
Important: Do not reboot the iPod while it says ***do not disconnect***, because that might have the bad effect that the modifications you made from your PC to the iPod might get lost. Make sure you eject the iPod so that it says *ready to disconnect* before you reboot it!
Also, do not reboot your iPod while it is booting for the first time **after a firmware update** (using the Apple updater). This will abort the ROM flashing, and, except for iPod from Nano 2G and Classic, this **WILL RENDER YOUR iPod USELESS**.
See also this Apple support document on resetting the iPod[2].

A.1.2 Disk Mode

Forces the iPod to wait for a Firewire/USB connection to a computer. If your iPod do not boot up, you can boot into Disk Mode and reload Apple firmware.
See also this Apple support document on disk mode[3].

A.1.3 Diagnostic Mode

Presents a menu of diagnostic utilities to check your iPod hardware. Use this menu to check if your iPod is working properly. Note the wheel does not work: you have to use the forward and rewind buttons to move around the menu.
For more information see Craig A. Finseth's article[4] on diagnostic mode. iPodlounge has an article on the iPod Photo's diagnostic mode[5].

[1] http://ipodlinux.org/wiki/Key_Combinations
[2] http://docs.info.apple.com/article.html?artnum=61705
[3] http://docs.info.apple.com/article.html?artnum=93651
[4] http://www.finseth.com/parts/ipod.php
[5] http://www.ipodlounge.com/index.php/articles/comments/ipod-photo-diagnostic-mode-revealed/

A.1.4 Device Firmware Upgrade Mode

DFU mode is a relatively new standard for upgrading firmware that is used in many devices like the OpenMoko and newer iPods.
DFU mode (since nano 2G) is stored in the on-processor bootrom. Newer iPods have both DFU mode and disk mode, while iPod Touch and iPhones have exclusively DFU mode.

The nano 2G has two DFU modes. One in the bootrom (and cannot be entered manually), and one stored in the NOR flash, inside norboot (and can be triggered through the key combination you can see below).

To switch to DFU mode on an iPod Nano 3G or 4G, follow these steps:

1. Make sure your iPod is turned on and connected to your computer.
2. Press the menu button and select (central) button simultaneously.
3. The iPod screen will go black, and the Apple logo will shortly appear (ie the iPod reboots).
4. Keep on pressing until the Apple logo turns into a black screen. This can take up to 10 seconds.
5. Release the menu and select buttons.

A.2 Combinations

A.2.1 1st, 2nd, and 3rd Generations

- Reboot iPod: Hold down the **Menu** and **Play** buttons.
  - You might have to hold both buttons for a very long time (∼30 seconds) in some cases.
  - If that still does not work, switch the "hold" button on and off and try again.
- Disk Mode: Immediately hold down the **Rewind** and **Fast Forward** buttons as the iPod reboots.[1]
- Diagnostic Mode: Hold **Rewind**, **Fast Forward**, and **Select** buttons as the iPod reboots.
- Third-part firmware (only when the *old* bootloader is installed): hold **Rewind** after rebooting the iPod

A.2.2 Mini, 4th, Photo, Nano, and 5G Generations

- Reboot iPod: Hold down the **Menu** and **Select** buttons.
  - You might have to hold both buttons for a very long time (∼30 seconds) in some cases.
  - If that still does not work, switch the "hold" button on and off and try again.
- Disk Mode: immediately hold down the **Play** and **Select** buttons as the iPod reboots.[1]
- Diagnostic Mode: hold **Rewind** and **Select** buttons as the iPod reboots.
- DFU Mode (NOR DFU): hold **Rewind** and **Play** buttons as the iPod reboots. (works on Nano 2G, but not on 5G, DFU protocol not explored yet)
- Third-part firmware (only when the *old* bootloader is installed): hold **Rewind** after rebooting the iPod. (Alternatively switch the hold button to "lock position" after rebooting)

The **Select** button is the one in the **center** of the click/scroll wheel.

[1] To reach disk mode, connect the iPod to your PC/Mac **before** starting. Then press and hold the 'Reboot iPod' combination above until the iPod reboots (you will notice a change on the screen). **Quickly** move your hand so you are now holding the 'Disk Mode' combination above. You have to do this before the Apple logo comes back (or maybe within 1 second afterwards).

A.2.3 Pictures

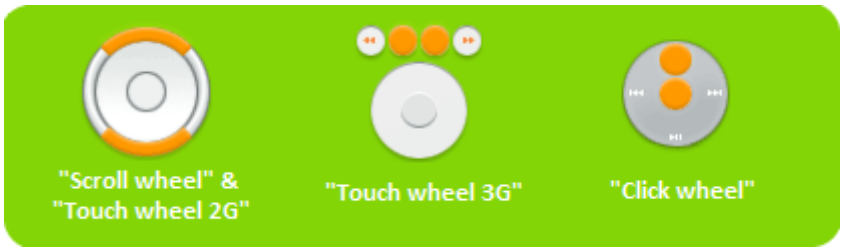

Fig. A.1: Keys for Reboot

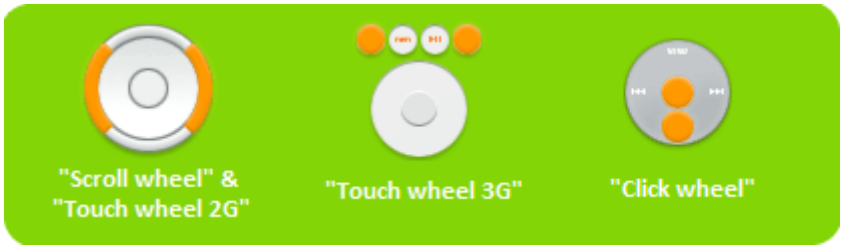

Fig. A.2: Keys for Disk Mode

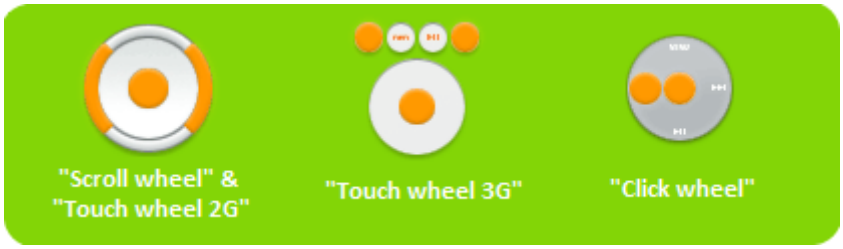

Fig. A.3: Keys for Diagnostic Mode

B iPods hardware specifications

Contents

B.1 Notes . . . 49
B.2 Fullsized iPods . . . 50
B.2.1 First Generation (1G) / Second Generation (2G) . . . 50
B.2.2 Third Generation (3G) . . . 50
B.2.3 Fourth Generation (4G) . . . 51
B.2.4 iPod Photo (Photo) / Color iPod (Color) . . . 51
B.2.5 Fifth Generation (5G) / Fifth Generation Enhanced (5.5G) . . . 52
B.2.6 Sixth Generation (6G/Classic) . . . 53
B.3 Small iPods . . . 53
B.3.1 iPod Mini (Mini1G) . . . 53
B.3.2 iPod Mini Second Generation (Mini2G) . . . 54
B.3.3 iPod Nano (Nano1G) . . . 54
B.3.4 iPod Nano Second Generation (Nano2G) . . . 55
B.3.5 iPod Nano Third Generation (Nano3G) . . . 55
B.3.6 iPod Nano Fourth Generation (Nano4G) . . . 56
B.3.7 iPod Nano Fifth Generation (Nano5G) . . . 56
B.4 Widescreen iPods . . . 57
B.4.1 iPod Touch . . . 57
B.5 Shuffle iPods . . . 57
B.5.1 iPod Shuffle . . . 57
B.5.2 iPod Shuffle Second Generation . . . 57
B.5.3 iPod Shuffle Third Generation . . . 57
B.5.4 iPod Shuffle Fourth Generation . . . 58
B.6 Links . . . 58

We would like to thank iPodLinux: this is a copy of the wiki "Generations"[1] page.

B.1 Notes

- Gestalt refers to the iPod's hardware revision number. This is found on the iPod's disk in the file "/iPod_Control/Device/SysInfo", listed under the key "boardHwSwInterfaceRev" (for the curious: this file does not exist on shuffles).
- Apple Name is the official name Apple uses to refer to each generation of iPod. This can be found in the description of the iPod updater in Software Update. Please keep semantics and capitalization verbatim.

[1] http://ipodlinux.org/wiki/Generations

B.2 Fullsized iPods

B.2.1 First Generation (1G) / Second Generation (2G)

The 'Scroll Wheel' and 'Touch Wheel' models.	
Distinguish	1G has white plastic around the top ports, and a rotating wheel. 2G has shiny metal around the top ports, and a stationary, "touch" wheel. Neither have a "dock" connector, but do have a Firewire connector. The 2nd gen iPods will have a cover on this connector.
CPU	Portal Player PP5002
Audio	Wolfson Microelectronics WM8721
I/O	FireWire: Texas Instruments TSB43AA82 PHYS and LINK
RAM	Samsung K4S561632C-TL1H (32 MB)
Storage	MK5002MAL (5GB) or MK1504GAL (10GB) 1.8" Hard Drive (1 platter, either 1 or 2 heads)
Display	Renesas HD66753 Graphics LCD Controller/Driver. 2 bit grayscale 160x128 LCD (Unknown supplier)
Power	MAX115 A2D for battery status Linear Technology LTC1731 a complete, constant-current/constant-voltage linear charge controller for fast charging of single-cell lithium-ion (Li-ion) batteries. Linear Technology LTC1726 combines the ability to monitor three supply voltages, at 61.5% threshold accuracy, with adjustable reset and watchdog functions.
Other	
Gestalt	0x00010000, 0x00010001, or 0x000100002= 1G; 0x00020000 or 0x00020001 = 2G
Apple Name	1G: iPod with scroll wheel; 2G: iPod with touch wheel

B.2.2 Third Generation (3G)

The 'Docking' model.	
Distinguish	Four buttons across between the touchwheel and the screen. First iPod with a dock connector.
CPU	Portal Player PP5002
Audio	Wolfson Microelectronics WM8731L Ole Wolff Buzzer OWMB-757526S
I/O	FireWire: Texas Instruments TSB43AA82 USB2: Cypress Semi CY7C68013-56LFC interface.
RAM	Samsung K4S561633C-R(B)L/N/P75. 32MB of SDRAM.
Storage	10GB, 15GB, 20GB, 30GB and 40GB - Sharp LH28F800BGHB-TTL90 Flash Memory
Display	Renesas HD66753 Graphics LCD Controller/Driver. 2 bit grayscale 160x128 LCD (Unknown supplier)
Power	Linear Technology LTC1733 Battery Charger Philips PCF50605HN PMU (Power Management Unit).
Other	Texas Instruments SN74CBTLV16211 Bus Switch
Gestalt	0x00030001 = 3G
Apple Name	iPod with dock connector
Notes	PCB Images

B.2.3 Fourth Generation (4G)

The 'Click Wheel' model. Also available in black with a red click wheel. (U2 model)	
Distinguish	Blueish hue to the screen. Buttons integrated in the new "clickwheel".
CPU	Portal Player PP5020
Audio	Wolfson Microelectronics WM8975 CODEC (should be similar to WM8971).
I/O	FireWire: Texas Instruments TSB41AB1 FireWire Physical Layer.
RAM	Hynix HY5V56D or SEC 434 XL75 SDRAM (32MB).
Storage	Toshiba MK2004GAL 20GB 1.8" HD / Toshiba MK4004GAH 40GB 1.8" HD
Display	Renesas HD66753 Graphics LCD Controller/Driver. 2 bit grayscale 160x128 LCD (Optrex and Wintek)
Power	Philips TEA1211, auto-up/down DC/DC converter circuit with I^2C-bus interface. Philips PCF50605HN PMU (Power Management Unit). SBW 340, surge protection.
Other	National Semiconductor LM3485 (marked S29B) Hysteretic PFET Buck Controller (Switching Regulator)
Gestalt	0x00050013 or 0x00050014 = 4G
Apple Name	iPod with Click Wheel
Notes	PCB Images

B.2.4 iPod Photo (Photo) / Color iPod (Color)

Color screen, click wheel; Also available in black with a red click wheel. (Color U2 model)	
Distinguish	Color screen. Cannot play movies with Apple's firmware.
CPU	Portal Player PP5020. (chip revision may differ)
Audio	Wolfson Microelectronics WM8975 CODEC (should be similar to WM8971).
I/O	FireWire: Texas Instruments TSB41AB1 FireWire Physical Layer. CCIR 601: Analog Devices ADV7179 NTSC/PAL Video Encoder
RAM	Hynix HY5V56D 32MB SDRAM.
Storage	Silicon Storage Technology SST39WF800A 8 Mbit Multi-Purpose Flash, Toshiba MK2006GAL HDD
Display	Renesas HD66789R LCD controller (in newer models only, original models unknown controller)
Power	Philips PCF50605HN PMU (Power Management Unit). National Semiconductor LM34910 Step Down Switching Regulator Philips TEA1211, auto-up/down DC/DC converter circuit with I^2C-bus interface.
Other	
Gestalt	0x00060000 = Photo; 0x00060004 = Color
Apple	Name Photo: iPod photo; Color: iPod with color display
Notes	16 bit color 220x176 2" LCD PCB Images

B.2.5 Fifth Generation (5G) / Fifth Generation Enhanced (5.5G)

Large color screen, thinner, plays MPEG and H.264 video.	
Distinguish	The front face has a glossy surface and is available in black and white. **60GB iPod**: This is a 5G (Fifth Generation) iPod only. **80GB iPod**: This is a 5.5G (Fifth Generation Enhanced) iPod only. **30GB iPod**: This can be either a 5G or 5.5G iPod. To distinguish the Fifth Generation iPod (Late 2006, 5.5G) from the original Fifth Generation (5G) iPod is by comparing the last three digits of the serial number. The 5.5G serial number's last three digits will be one of the following: V9K, V9P, V9M, V9R, V9L, V9N, V9Q, V9S, WU9, WUA, WUB, WUC, W9N and X3N. The Fifth Generation U2 Special Edition iPod (30 GB Late 2006) serial number's last three digits are W9G. Also, if your iPod has a search feature, it is a 5.5G.
CPU	Portal Player Portal Player PP5021C-TDF (like PP5020) The PP chip presents itself as PP5022C in software.
Audio	Wolfson Microelectronics WM8758 Audio codec
I/O	CCIR 601: Broadcom BCM2722 Video decoder/processor
RAM	Samsung K4S56163PF SDRAM - 32MB (in the 30GB model - like previous models, 60GB model has 64MB with Samsung part number K4M51163PC.)
Storage	**5G**: Toshiba MK3008GAL 30GB 1.8" HDD / Toshiba MK6008GAH 60GB 1.8" HDD **5.5G**: Toshiba MK3008GAL 30GB 1.8" HDD / Toshiba MK8010GAH 80GB 1.8" HDD (it should be an Apple branded MK8009GAH) Silicon Storage Technology SST39WF800A 8 Mbit Multi-Purpose Flash
Display	Unknown LCD Controller 16 bit color 320x240 2.5" LCD (Manufacturer: Toshiba-Matsushi ta markings: 1WX510015194)
Power	Philips PCF50607 PMU (Power Management Unit) Linear Technology LTC4066 Linear USB Power Manager / Li-Ion Battery Charger National Semiconductor LM34910 Step Down Switching Regulator Philips TEA1211 DC/DC converter with I2C
Other	Cypress CY8C21434 PSoC Mixed Signal Controller - Touch Pad
Gestalt	0x000B0005 or 0x000B0010 = 5G
Apple Name	iPod with video
Notes	Vivisection (PCB Pictures) Another picture of 5.5G PCB

B.2.6 Sixth Generation (6G/Classic)

6th Generation iPod, Metal front face.	
Distinguish	Metal front face with a sharp edge around the perimeter. The surface of the metal front face is abrasive/matted. Available in black or silver.
CPU	Samsung S5L8702 SoC, which contains an ARM926EJ-S processor and 256KB of internal SRAM.
Audio	Apple Branded Cirrus Chip, Model Number Cannot Be Made Out in iFixit Teardown
I/O	??
RAM	64MB of external SDRAM
Storage	80GB or 160GB 1,8" harddrive, 80GB: Toshiba MK8022GAA / 160GB: Toshiba MK1626GCB
Display	2,5" 320x240 color LCD
Power	??
Other	??
Gestalt	??
Apple Name	iPod classic
Notes	iFixit Teardown

B.3 Small iPods

B.3.1 iPod Mini (Mini1G)

Ultra-portable hard drive-based MP3 player.	
Distinguish	Small unit, metal case. Available in muted, pastel metallic colors. Click wheel font is grey and does not match the color of the iPods case. Hard drive noise may be heard from the device.
CPU	Portal Player PP5020 The PP chip presents itself as PP5020E in software.
Audio	Wolfson Microelectronics WM8731L
I/O	FireWire: Texas Instruments TSB41AB1 FireWire Physical Layer.
RAM	Samsung K4S561633C-R(B)L/N/P75. 32MB of SDRAM.
Storage	Silicon Storage Technology SST39VF800A 8Mbit Flash.
Display	Renesas HD66753 Graphics LCD Controller/Driver. 2 bit grayscale 138x110 LCD
Power	Philips PCF50605HN PMU (Power Management Unit). Linear Technology LTC4055 USB Power Controller and Li-Ion Linear Charger Texas Instruments TPS62046 step-down dc-dc converter National Semiconductor LM3485 Hysteretic PFET Buck Controller (Switching Regulator)
Other	
Gestalt	0x00040013 = 1G Mini
Apple Name	iPod mini

B.3.2 iPod Mini Second Generation (Mini2G)

Second generation iPod mini.	
Distinguish	Small unit, metal case. Capacity is engraved on the back. Available in vibrant metallic colors. Click wheel font matches the color of the iPods case. Better battery life (18hrs). Hard drive noise may be heard from the device.
CPU	Portal Player PP5022 processor (Dual 80 MHz ARM 7TDMI processors) The PP chip presents itself as PP5022B in software.
Audio	
I/O	
RAM	32MB of SDRAM
Storage	4GB / 6GB Hitachi 3600 RPM ATA-66 Micro drive (internal specifications: 4GB and 6GB)
Display	Renesas HD66753 Graphics LCD Controller/Driver. 2 bit grayscale 138x110 LCD
Power	
Other	
Gestalt	0x00070002 = 2G Mini
Apple Name	iPod mini Second Generation

B.3.3 iPod Nano (Nano1G)

Tiny Flash-based color iPod.	
Distinguish	Black or White plastic front half with a shiny metal back half.
CPU	Portal Player PP5021C-TDF The PP chip presents itself as PP5022C in software.
Audio	Wolfson Microelectronics WM8975G (DAC)
I/O	
RAM	SEC 534 BG75 K4M56163PG - Samsung 4M x 16Bit x 4 Banks Mobile SDRAM
Storage	Samsung 534-K9WAG08U1M - Storage flash Silicon Storage Technology 55LD019K 4MB - Firmware Flash
Display	Renesas HD66789R compatible LCD Controller 16 bit color 176x132 1.5” LCD (three manufacturers Optrex, Sharp Electronics, and Toshiba-Matsushita; edge markings: 4BN8-116, 045JKHN7, NWP74 MA50A111)
Power	Philips CF50607 PMU (Power Management Unit). National Semiconductor JM54RM LM34910 Step Down Switching Regulator
Other	Cypress CY8C21434 - Click wheel controller
Apple Name	iPod Nano
Notes	dissection images(00-35) Overview, opened Back and Front, Front, Storage PCB, PCB, Accu, PCB front, PCB Front detail, Storage, Storage removed, Display

B.3.4 iPod Nano Second Generation (Nano2G)

Tiny iPod with Color Display	
Distinguish	Metal case which wraps around the entire iPod. Available in a few colors including silver and black.
CPU	S5L8701 SoC, which contains an ARM940T processor and 176KB of internal SRAM.
Audio	Wolfson WM8750S, based on the WM8975G)
I/O	
RAM	Samsung 32MB (256 Megabits) SDRAM Silicon Storage Technology SST39WF800A 8 Mbit Multi-Purpose Flash
Storage	Provided by either Samsung ,Hynix or Toshiba depending on the model
Display	Sharp/Optrex/Toshiba 1.5-inch (diagonal) liquid crystal display with blue-white LED backlight 176-by-132-pixel resolution, .168-mm dot pitch
Power	Apple branded PMU (Probably a Philips PCF50635, based on the PCF50607) National Semiconductor LM34910 Step Down Switching Regulator
Other	Cypress CY8C21434 - Click wheel controller Linear Technology LTC4066 USB battery charging circuitry
Gestalt	Does not exist in new Firmware.
Apple Name	iPod Nano
Notes	Ars Technica Dissection and Analysis

B.3.5 iPod Nano Third Generation (Nano3G)

Smaller, metal case, larger screen, plays videos	
Distinguish	Shorter, stubbier, large "TV shaped" color screen. Can play movies with Apple firmware. Sharp edge around the front metal face
CPU	S5L8702 SoC, which contains an ARM926EJ-S processor and 256KB of internal SRAM.
Audio	Apple branded 76BZKTM (Known to be some Wolfson Chip), or is it the Apple branded 338S0462 76ARKTM chip (it is on the same side as the CPU but right next to the output capacitors on the opposite side)?
I/O	Touch wheel uses a Synaptics controller T5 0726 ACV1846 (ribbon and wheel also Synaptics branded, 940-151-4RA)
RAM	Hynix HYE1SM256 - 32MB SDRAM or Samsung K4X56163PI-GGC3 - 32MB Mobile DDR SDRAM
Storage	Intel, Micron or Samsung Chip, 4GB: Intel 29F32G08FAMB2 / 8GB: Samsung 728-K9HCG08U5M
Display	320x240, 204ppi 2" color LCD screen
Power	Apple branded 338S0408 07288HAX (e4 in a circle like on the Intel NAND flash)?
Other	??
Gestalt	??
Apple Name	iPod nano
Notes	iFixit Teardown

B.3.6 iPod Nano Fourth Generation (Nano4G)

Slimmer, rounded metal case, longer screen, plays videos sideways	
Distinguish	Portrait color screen, rounded edges. Plays movies horizontally. Has the Genius feature. Built-in accelerometer.
CPU	S5L8720 SoC containing an ARM1136JZF-S processor and 192KB of internal SRAM.
Audio	??
I/O	??
RAM	32MB of external SDRAM
Storage	Toshiba TH58NVG6D1DLA87 4GB Flash RAM chips
Display	2 inch, 320x240 resolution, 204ppi (Unknown supplier and unknown controller)
Power	??
Other	??
Gestalt	??
Apple Name	iPod nano
Notes	iFixit Teardown

B.3.7 iPod Nano Fifth Generation (Nano5G)

Added video camera, radio	
Distinguish	Bigger screen, built-in video camera, radio.
CPU	Apple branded Samsung S5L8730 ARM CPU
Audio	??
I/O	??
RAM	??
Storage	Toshiba NAND
Display	2,2 inch, 240x376 resolution (Unknown supplier and unknown controller)
Power	??
Other	??
Gestalt	??
Apple Name	iPod nano
Notes	Teardown CPU model similar to the one in Touch2G (S5L8720). IPSW files have a header "87302.0". It seems that 5G uses a derivation of iPhone 8900 files

B.4 Widescreen iPods

B.4.1 iPod Touch

iPod with touch screen	
Distinguish	Touch screen, no click wheel.
CPU	Apple branded ARM cpu, 339S0029ARM with integrated Samsung memory
Audio	Wolfson Microelectronics WM87588G Audio codec
I/O	??
RAM	128MB
Storage	Toshiba TH58NVG5D4CTG20 4GB Flash RAM chips for 8GB,16GB or 32GB Flash-based storage
Display	3,5" 480x320 widescreen multi-touch display
Wireless	LAN Marvell W8686B13 (802.11a/b/g WLAN)
Power	??
Other	??
Gestalt	??
Notes	iFixit Teardown 2D barcodes reads 9C73608WAX45B (unable to read the one on the battery). More details on this iPod's hardware are currently unknown and will be added whenever someone does a vivisection.

B.5 Shuffle iPods

B.5.1 iPod Shuffle

Ultra-portable flash-based iPod. Listed here for completeness.	
Distinguish	Size similar to a pack of gum.
CPU	SigmaTel's D-Major. This chip has a DSP56004 core. There is currently no recent GCC compiler for DSP56000 and therefore no uClinux/iPodLinux.
Gestalt	Does not exist on iPod shuffles.
Apple Name	iPod shuffle
Notes	Old gcc DSP56000 compiler

B.5.2 iPod Shuffle Second Generation

Ultra-portable flash-based iPod. Listed here for completeness.	
Distinguish	Clip-on design.
CPU	Apple branded ARM chip, 337S3300 844A N05WDK01 0642
Gestalt	Does not exist on iPod shuffles.
Apple Name	iPod shuffle

B.5.3 iPod Shuffle Third Generation

Apple calls this also second generation Shuffle. The only difference in naming is:

- iPod shuffle (2nd generation Early 2008) (introduced 2008-02)
- iPod shuffle (2nd generation) (introduced 2006-09)

Ultra-portable flash-based iPod. Listed here for completeness. (Same as 2nd Generation)	
Distinguish	Clip-on design.
CPU	??
Storage	1GB or 2GB
Gestalt	Does not exist on iPod shuffles.
Apple Name	iPod shuffle

B.5.4 iPod Shuffle Fourth Generation

Please note that Apple calls this the third generation Shuffle.

Ultra-portable flash-based iPod. Listed here for completeness.	
Distinguish	Clip-on design. As small as a key.
CPU	??
Storage	4GB only
Gestalt	Does not exist on iPod shuffles.
Apple Name	iPod shuffle
Notes	Has got voice function installed, as user interface.

B.6 Links

iLounge has written a great article, "A Brief History of iPod": http://www.ilounge.com/index.php/articles/comments/instant-expert-a-brief-history-of-ipod/.
Apple has a nice guide about identifying different iPod models: http://support.apple.com/kb/HT1353.

Credits

Special thanks to :

- Our teacher *Emmanuel Fleury* without whom this report would not have existed
- Michael 'TheSeven' Sparmann who highly participated to the project and had the patience to listen to our questions and give us some of his time to explain us many points
- Christophe 'tof' Riehl, Cory 'cmwslw' Walker, Dave 'linuxstb' Chapman, Taylor 'n00b81' Gordon, Felix 'fxb' Bruns, David 'planetbeing' Wang, Finn 'Farthen' Wilke and Franz-Josef 'liar' Haider, who highly contributed as well to achieve the whole understanding of the iPod nano 2G and the port of a third-pard firmware on it
- Everybody else who donated broken or even working hardware

Here is also a non-exhaustive list of those who contributed to the project/report.
Please excuse us if some people are missing :

3mpty
A W
Aaron P. D'Souza
Alex Willisson
Alexander Jørgensen
Alexander Papst
Amaya
Anton Zweistein
Apage43
Ari
Bahattin Tozyilmaz
Benoit Badrignans
Benoît Ryder
Biscuit Thomas
bizthepirate
C@che
DaHoC
Dan Andrews
Daniel Mason
EdwardRF
GearForce
Isaac Caldwell
James Watkins
Jean-Damien Brossillon
Jean-Loup Le Roux
Jeremy Prater
Jérôme Soumagne
John Pitsburg
John Gaylord
Keenan Pepper
Keripo
lala32
Manuel Naranjo
Markus Macher
Martin Sandsmark
mat h
max
Michael Thomas
ndsrox
Niklas Ulvinge
Oliver McFadden
Paul Martin
Paul McCabe
Patrick Wohlschlegel
Raoul Guggenheim
Sebastian Elsner
Sebastian Schutte
Sébastien Fourestier
serpilliere
Stephane Dekeyzer
Tom Stellard
Tyler Steinmetz
Vincent Mauge
Wayfarer
William Poetra Yoga Hadisoeseno

And of course all of those who participated to the RockBox project:
http://svn.rockbox.org/viewvc.cgi/trunk/docs/CREDITS