HomePlugAV PLC: Practical attacks and backdooring

Sébastien Dudek
19/10/2014

SOGETI

Who am I

Sébastien Dudek (@FIUxIuS)

- Has joined the ESEC R&D lab in 2012 after his internship (subject: Attacking the GSM Protocol Stack)
- Interests: radiocommunications (WiFi, RFID, GSM, PLC...), network, web, and Linux security.
- My story with PLCs:
  - moved out to a shared apartment;
  - angry with my room mate's WiFi (obstacles, perturbations...) $\rightarrow$ PLCs are cheap and could solve my problem;
  - and I've wanted to learn more about these little devices...

Summary

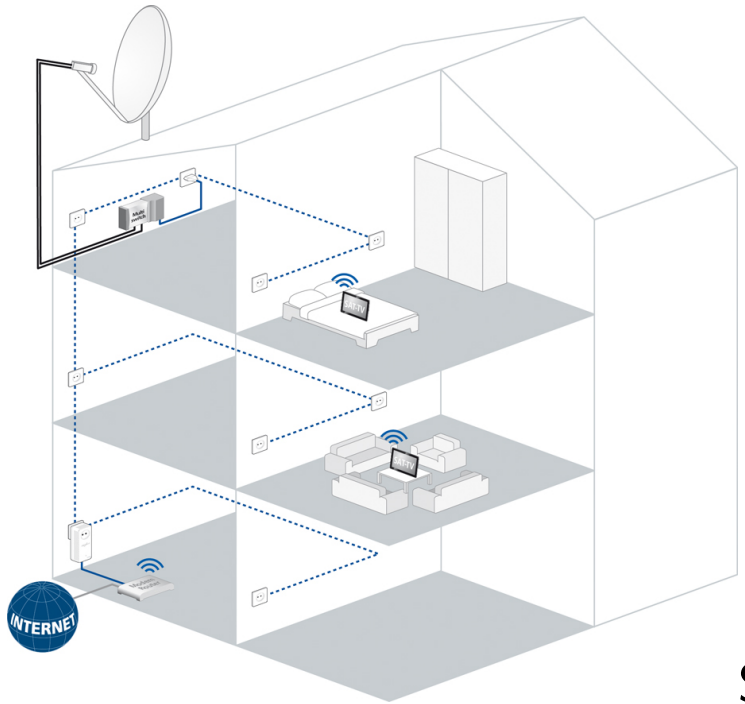
Introduction

- PLC: Powerline Communication $\neq$ Programmable Logic Controller (known on SCADA and other Apocalypse things...)
- Principle discovered by Edward Davy in 1838
- Released in the early 2000s for home applications
- Evolves a lot in term of speed
- Other systems like Cenélec (3-148.5 kHz low voltage) are used : meter readings, intruder alarms, fire detection, gaz leak detection, and so on

But how does it looks like at home?

PLC at home

The following pictures shows a house equipped with PLC devices:

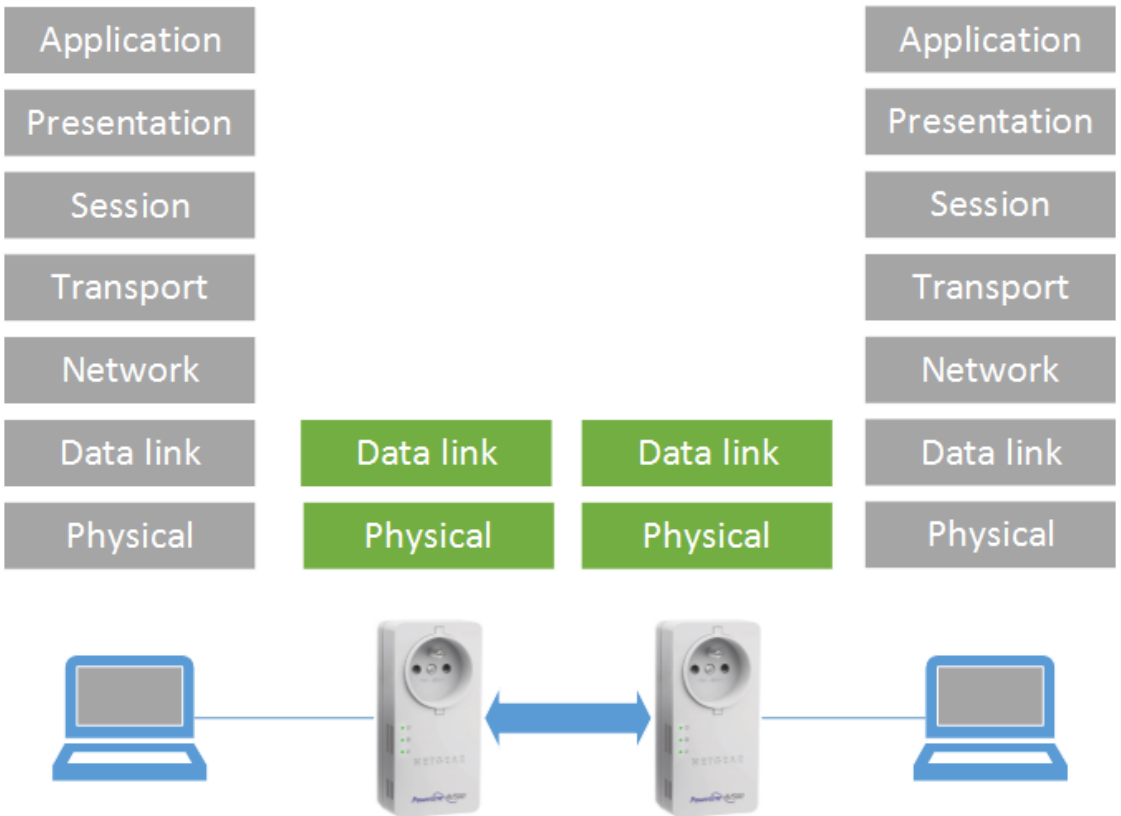


Source: devolo

Only one PLC is connected to internet and distributes it to other PLC → a user shouldn't worry about it's network topology.

PLC layers

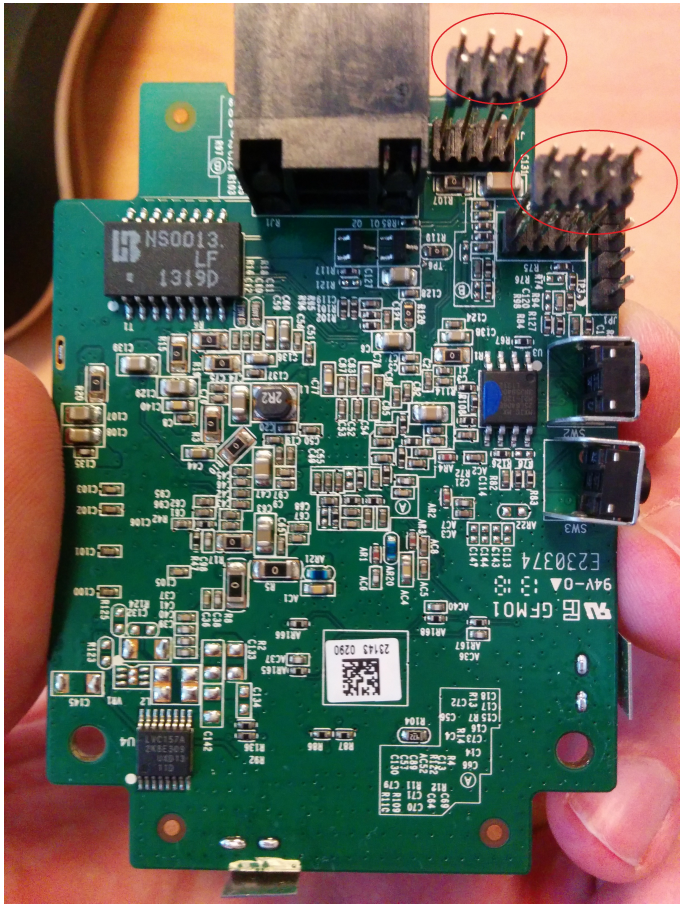
A PLC uses layer 1 and 2 of the OSI model ⇒ IEEE 802.3



Collision avoidance

- Use of CSMA/CA (Carrier Sence Multiple Access/Collision Avoidance)
- TDMA → allocate a period of transmission time for each station
- 1 TDMA frame used for CSMA/CA frames that don't need QoS

The hardware: divided in two parts



Vendor part

PLC part

Communications

Computer ↔ PLC

- Communicate through Ethernet on MAC layer
- Clear text (no ciphering)

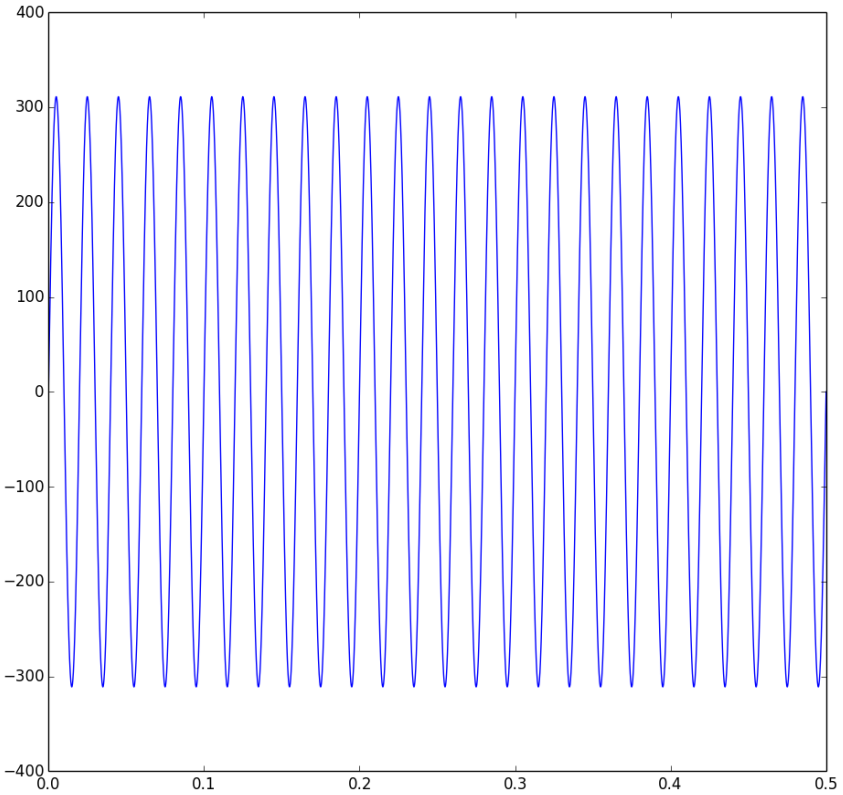
PLC ↔ PLC

- Communicate through powerline
- Data is encrypted (using AES CBC 128 bits on new PLCs)

Electrical properties: the power-line

AC voltage

- AC voltage at 50 Hz $\rightarrow$ signal do 50 cycles/s
- Could be represented by the formula:
  $Ps = A\sqrt{2}\sin(2\pi ft)$



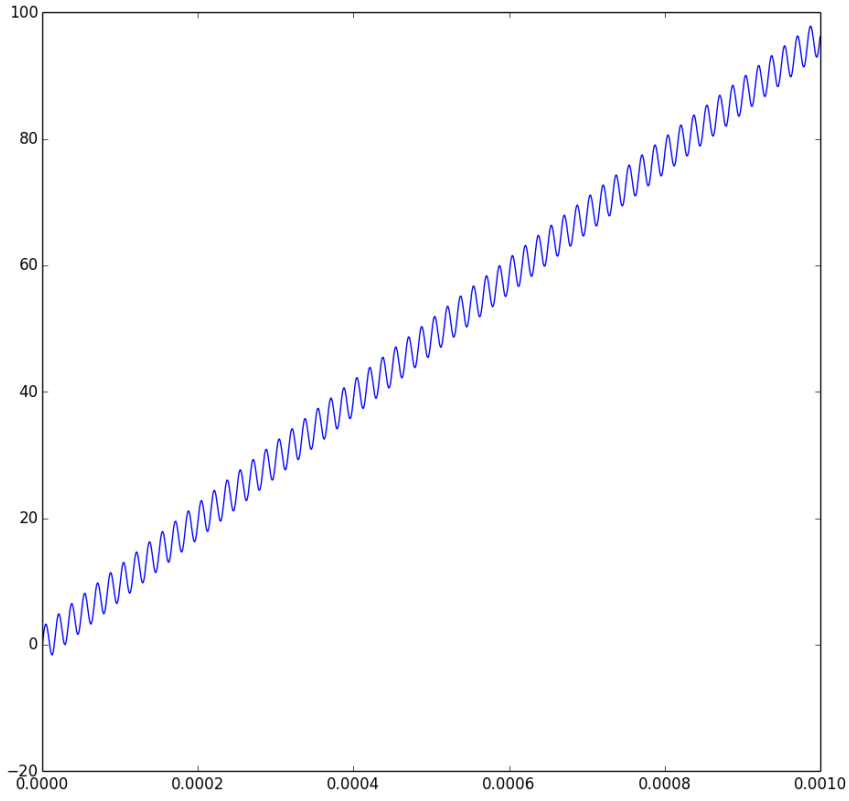
A is 220V in Europe, or 100V in US/Japon, f the number of cycles/sec (50 Hz in Europe for example).

Electrical properties: adding our signal

To transport our data on electrical power we use superposition:

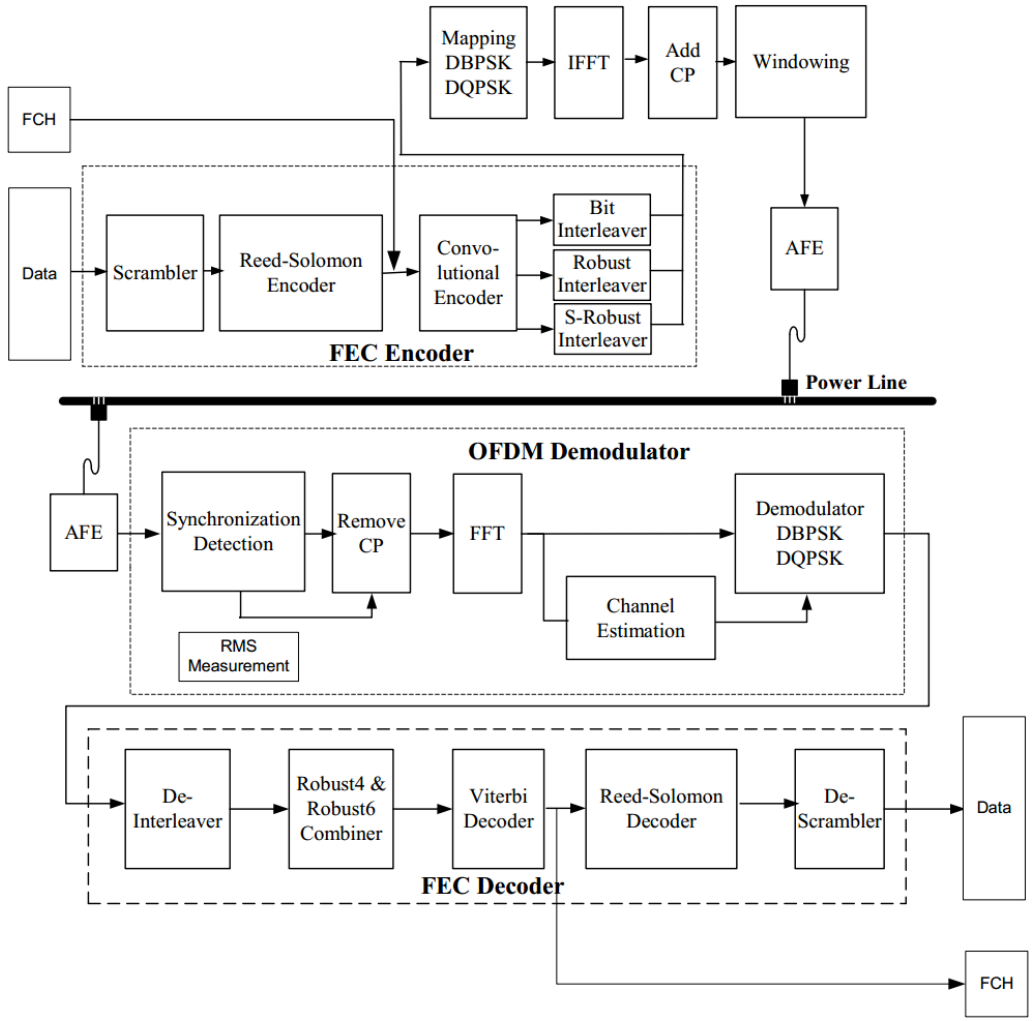
Superposition

- Suppose the carrier is 60 kHz:
  $Ca = 2\sqrt{2}\sin(2\pi 60000t)$
- Sum the power supply with the carrier:
  $Ps + Ca = 220\sqrt{2}\sin(2\pi 50t) + 2\sqrt{2}\sin(2\pi 60000t)$



But we need error detection, code mapping and multi-carrier modulation!

Digital Signal Processing (DSP)



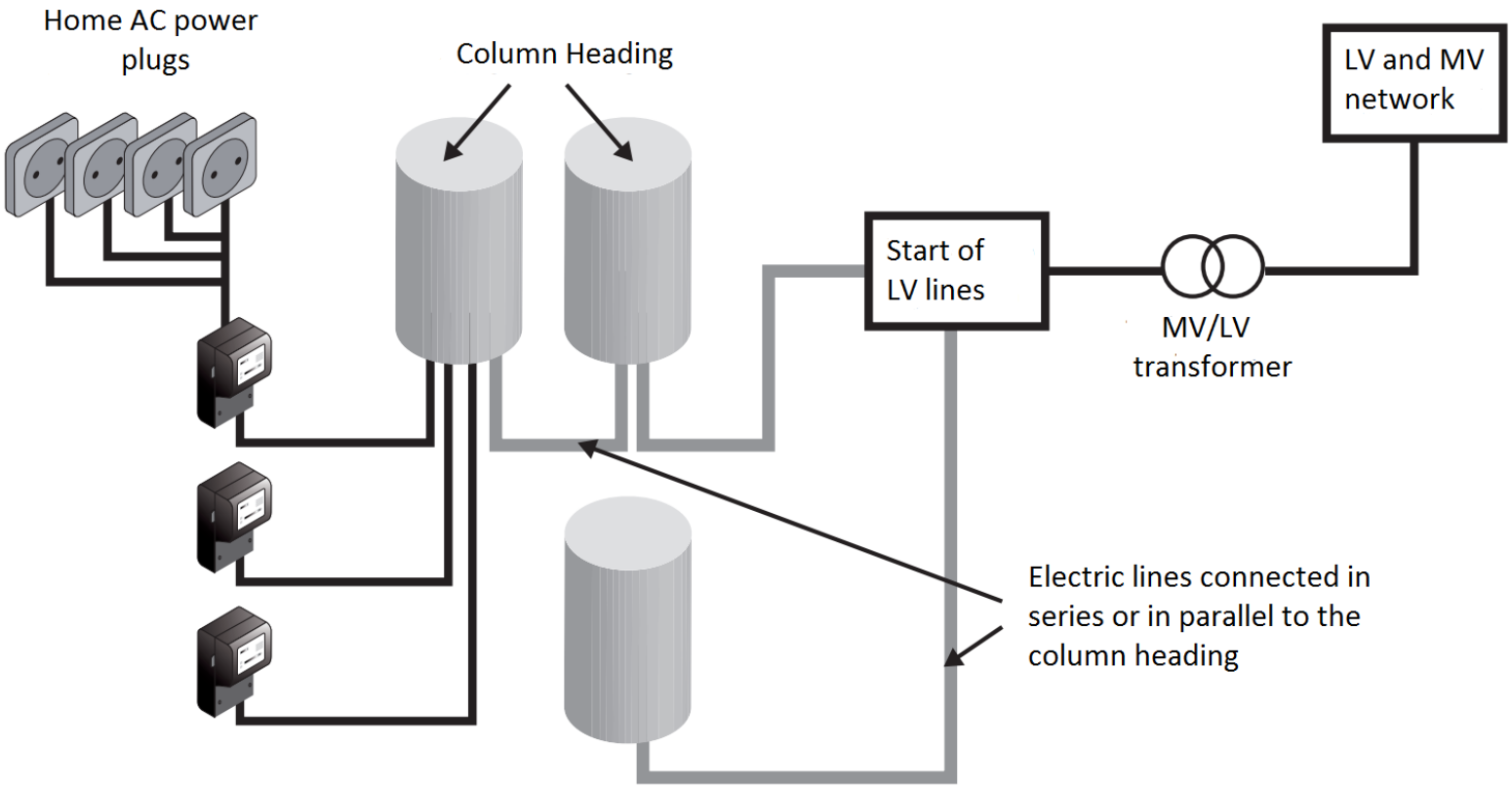
sourcce: G3-PLC

Steps in brief

1. data scrambling;
2. turbo encoding;
3. modulation of control and data frames;
4. form OFDM symbols by constellation;
5. windowing.

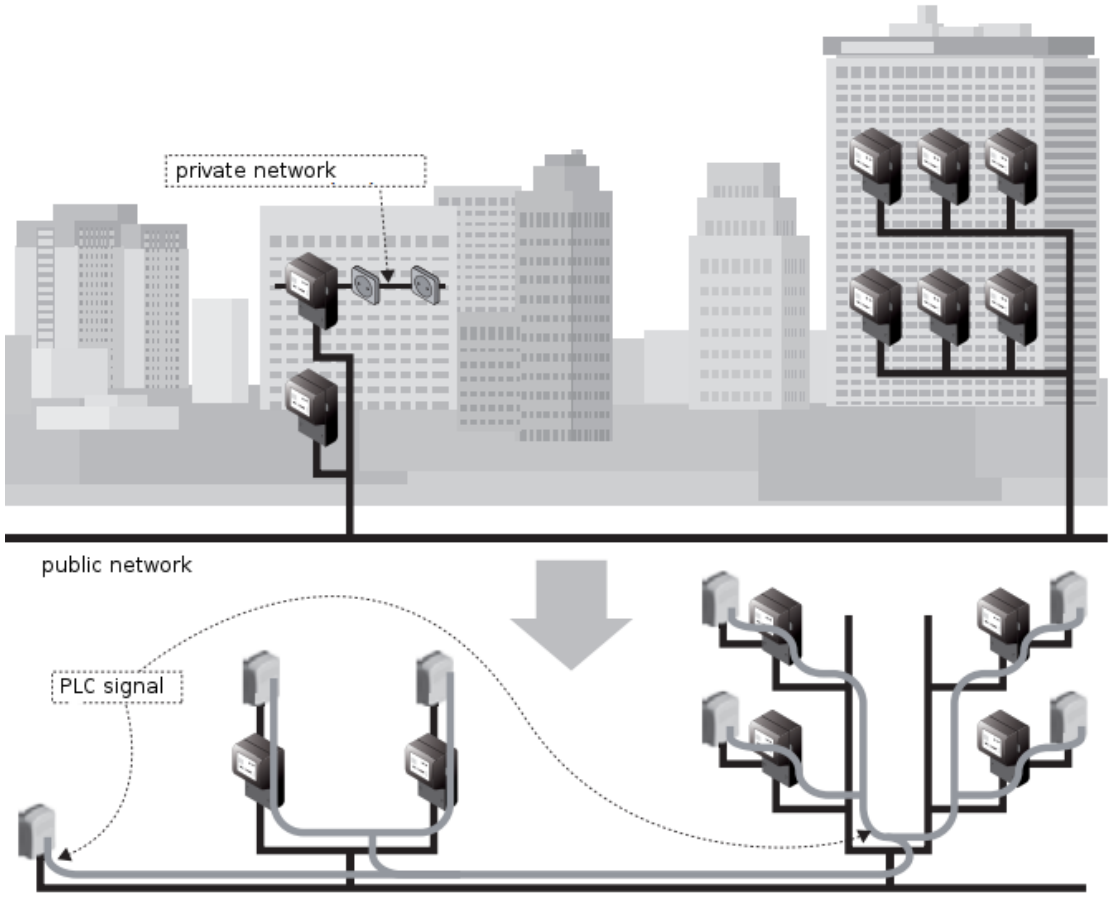
Electrical network

In france, the distribution network is similar to the telephony network (RTC)



source: PLC in Practice by Xavier Carcelle

Public and private network: myths and reality



source: PLC in Practice by Xavier Carcelle

Myth

Counters restrict PLC data spreading.

Reality

- No choc-coil → we can communicate:
  - from one appartment to another;
  - from the building lobby to someone's flate (3rd and 4th floor).

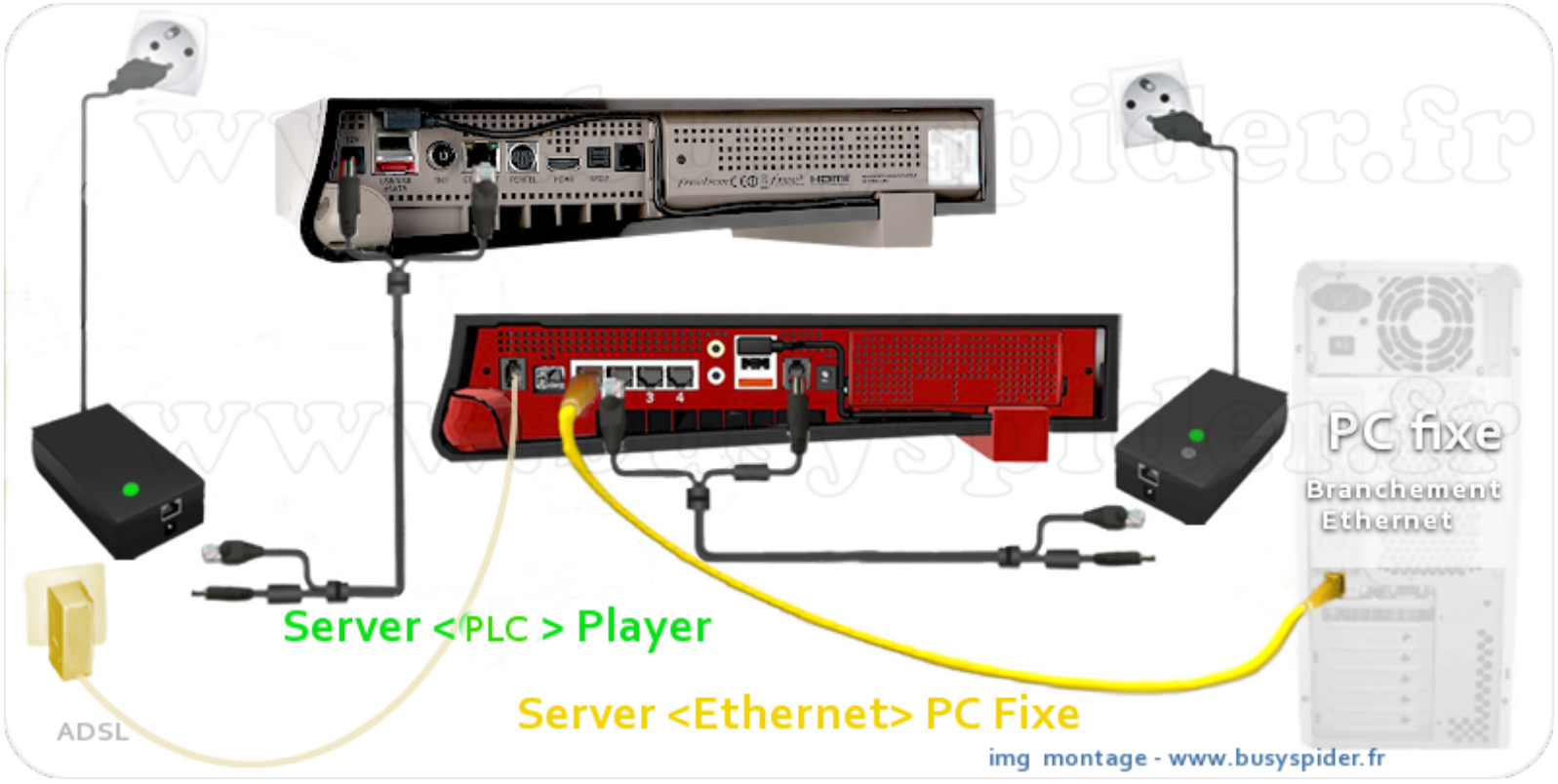
Old choc-coils are mostly ineffective to block MF/HF frequencies.

Our devices:

Model	Max Speed	Chipset	Extra features
XAV5401	500 Mb/s	Qualcomm Atheros 7420	
XWN5001	500 Mb/s	Qualcomm Atheros 7420	Smart Plug + WiFi N300
TL-PA6030	600 Mb/s	Qualcomm Atheros 7450	
FreeplugV1	200 Mb/s	INT6300	
FreeplugV2	200 Mb/s	INT6400	

PLCs embedded in power supply: example with Freeplugs

- An ethernet cable is joined with the power supply cable
- Normally, a "default" user will connect everything → just to be sure that everything will work fine...



Summary

Publications

- Power Line Communications in Practice by Xavier Carcelle → a must read!
- HomePlug AV Security Mechanisms by Richard Newman, Larry Younge, Sherman Gavette, and Ross Anderson, published in 2007
- MISC #37 HomePlug Security by Xavier Carcelle
- HomePlug Security by Axel Puppe and Jeroen Vanderauwera → gives an otherview of key bruteforcing for old devices

These publications give an overview of HomePlug security mechanisms. But just one paper really focuses on possible and pratical attacks...

Tools

- plconfig → manage PLCs over the network
- FAIFA by Xavier Carcelle (similar to plconfig)
- Vendors software (that we used at first)
- Wireshark has a dissector for HomePlugAV

But no scapy Layer exists for HomePlugAV to mess with the HomePlugAV protocol.

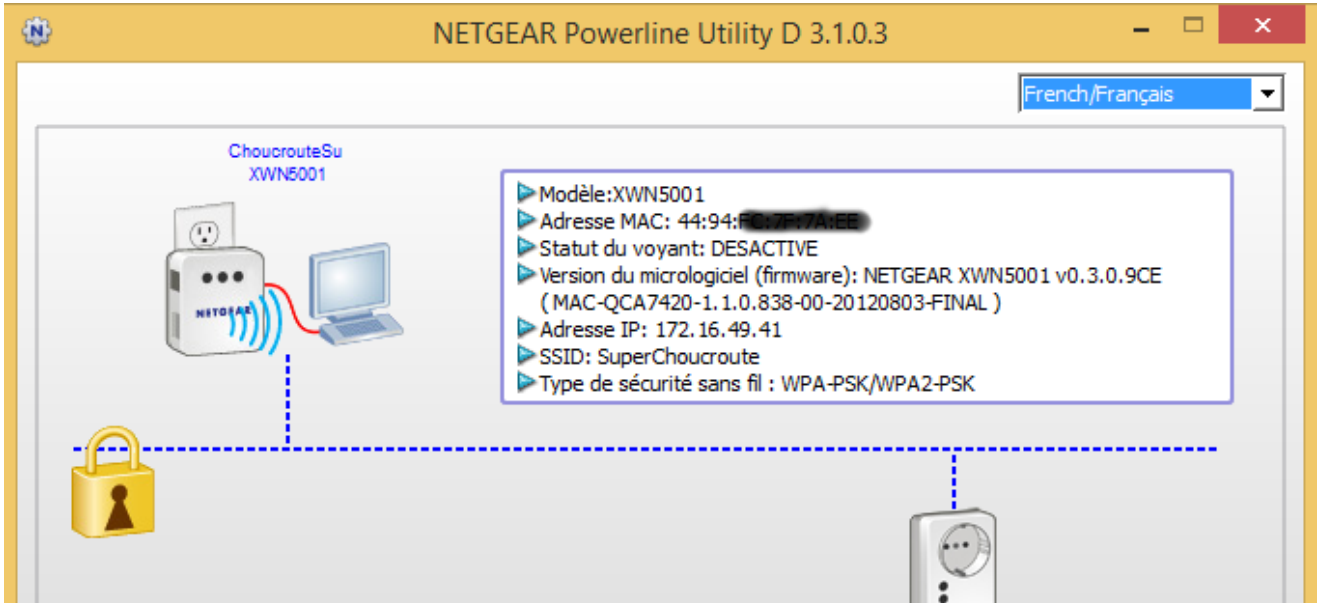
Summary

Vendors utility: example with Netgear

3 different ways to configure our PLC network

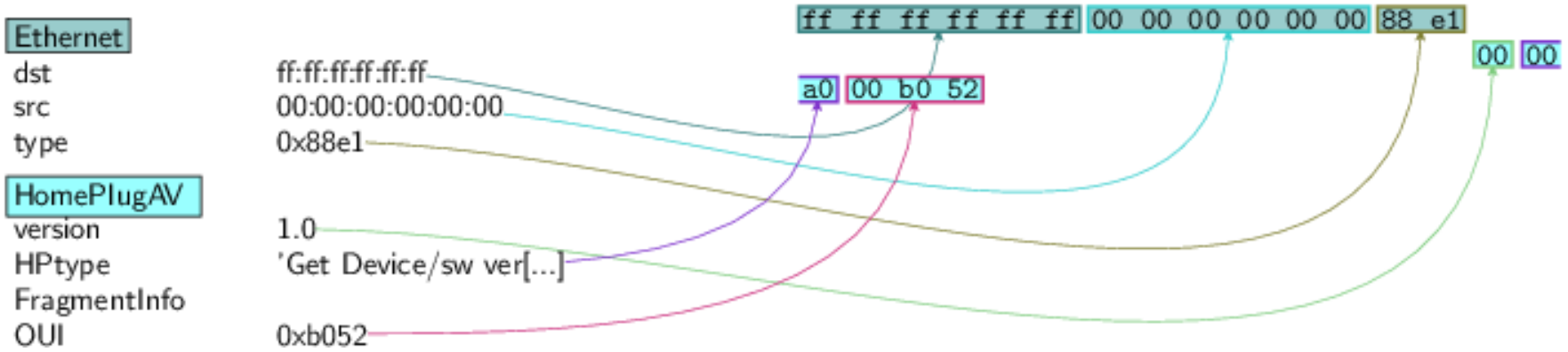
- default configuration (open network/default key);
- pairing button (easy way);
- or with a custom key (paranoid way → our case).

The software retrieves PLC information as follows:



Analysis with our scapy Layer: Device Type message

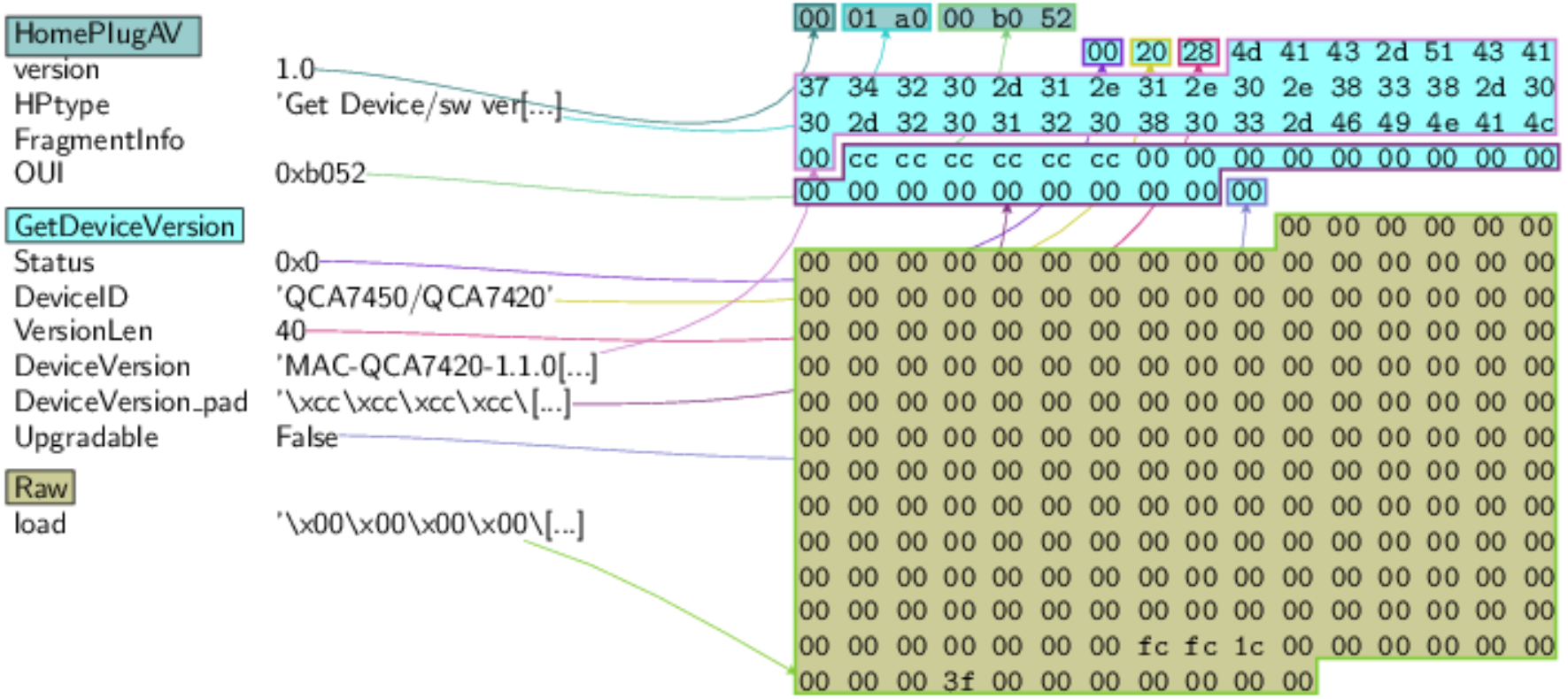
To retrieve devices type, the software broadcasts a "Get Device Type Request".



The software uses a Atheros broadcast address, but just to be sure it will work with all devices (INTELLON, Atheros, Qualcomm...), we can broadcast it with `ff:ff:ff:ff:ff:ff` address.

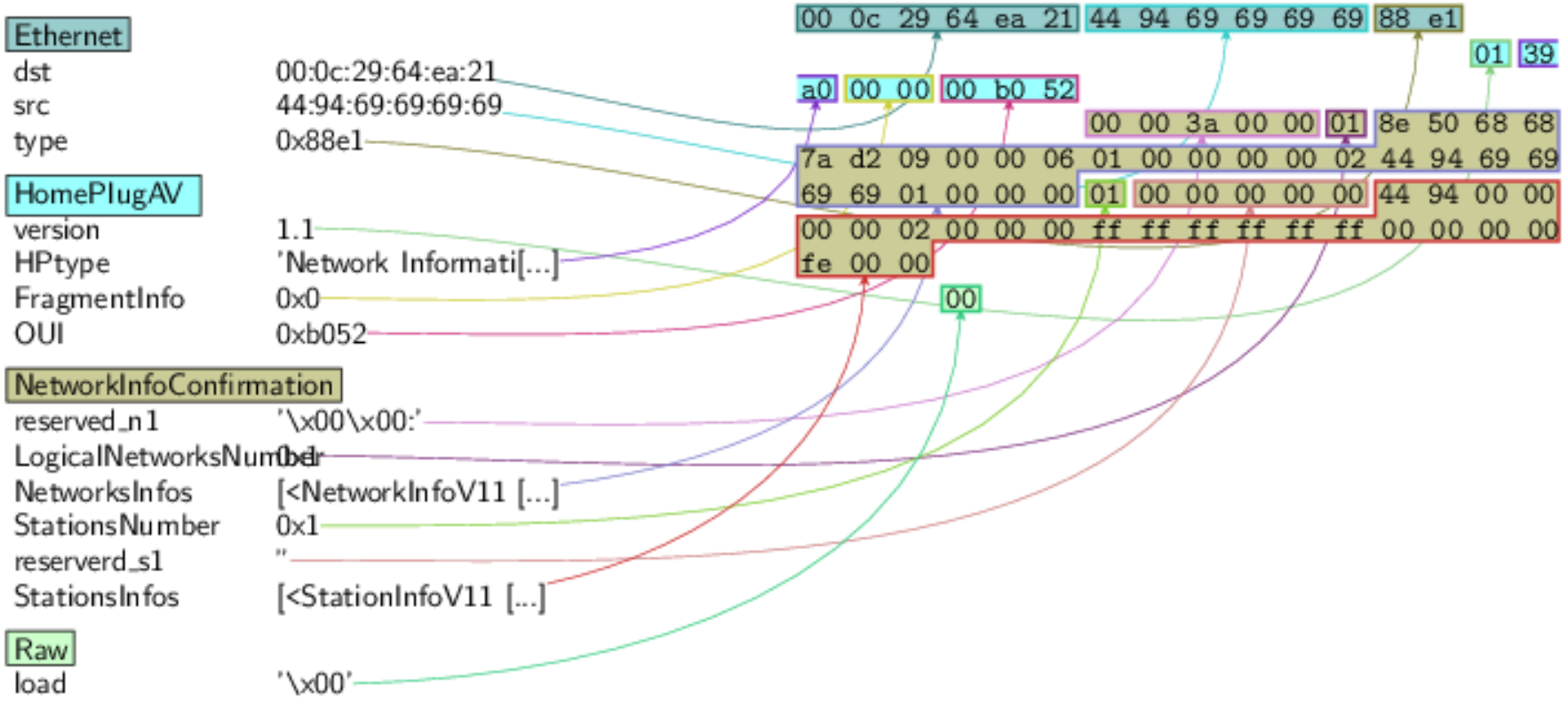
Device Type message: the confirmation

If the type request exists, you get a confirmation message with a "Status" field (0x0 = Success) followed with data:



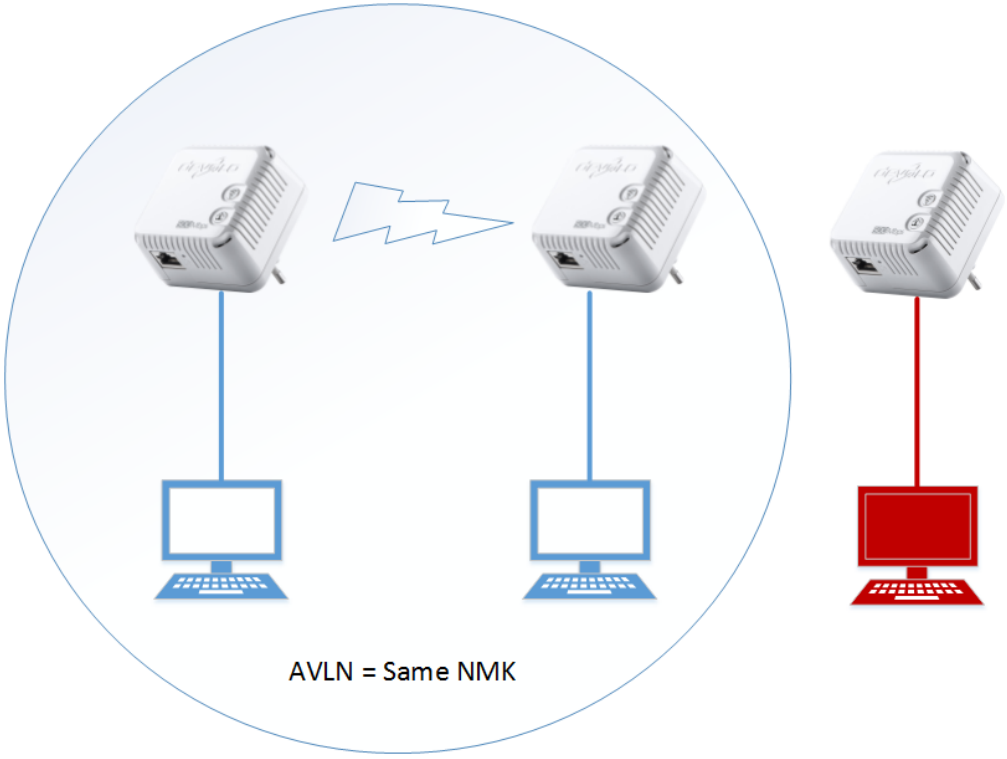
Network information

To get information about the CCo (Central Coordinator) and stations connected, the software send a "Network Information Request → then we get a"Network Information Confirmation" packet.



A typical PLC network

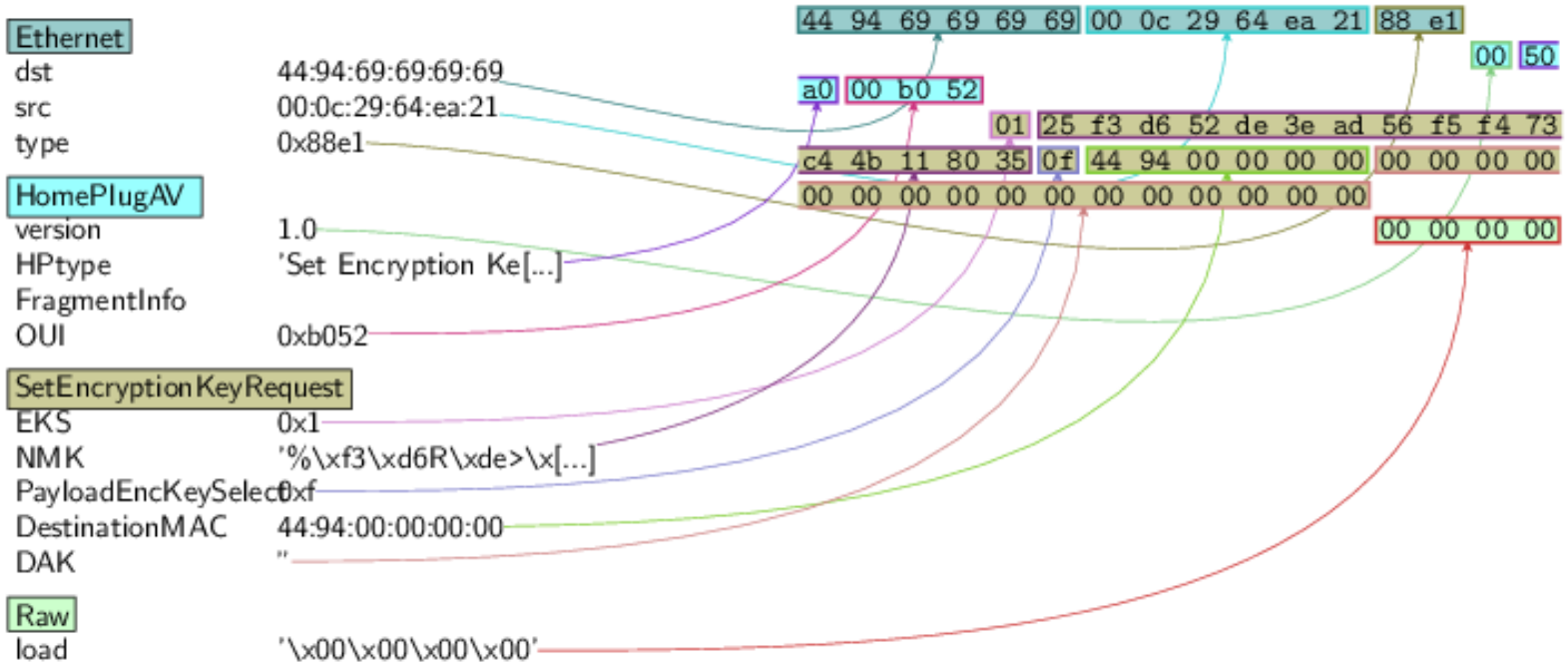
- The CCo manages contention-free streams time allocation, period for CSMA access + defines a AVLN node
- We can talk with other PLC of the same AVLN



The software can change the NMK passphrase, sending it to the targeted PLC.

Change the passphrase: SetEncryptionKeyRequest

We change local device's NMK passphrase:



Remotely

In remote, we need to precise a DAK (Direct Access Key) to change the NMK (Network Membership Key). This could be interesting...

NMK and DAK generation

- The NMK and DAK keys are generated the same way
- They use the Password-Based Derivation Function 1 (PBKDF1):
  - DAK or NMK$= PBKDF1(P, S, HF, c, dkLen)$;
  - $P \rightarrow$ the passphrase;
  - $S \rightarrow$ the salt;
  - $HF \rightarrow$ the hash function;
  - $c \rightarrow$ the number of iterations;
  - $dkLen \rightarrow$ the digest key length.
- The main parameters are known:
  - $S = 0x08856DAF7CF58185$ for DAK, $S = 0x08856DAF7CF58186$ for NMK;
  - HF is SHA-256;
  - $c = 1000$;
  - $dkLen = 16$ (bytes).

Attacks on NMK

Interception

1. Listen for broatcasted packets, MITM the administrator or fake the MAC address
2. and sniff the "Set Key Encryption Key" packet

LAN attack

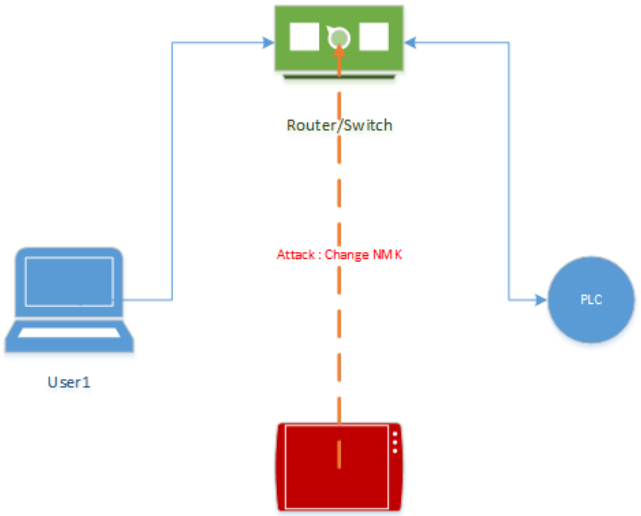
Bruteforce the NMK

Attacks on NMK

Interception

LAN attack

- a local device can be configured without any DAK
- But also: every device is connected to a switch/router are considered as local device in the network (don't need DAK).



Attacks on NMK

Interception

LAN attack

Bruteforce the NMK

1. Bruteforce the NMK from a dictionnary;
2. Change local device NMK by the interated one;
3. Send discovery packet to see if we joined any network.

Attacks on NMK

Interception

LAN attack

Bruteforce the NMK

1. Bruteforce the NMK from a dictionnary;
2. Change local device NMK by the interated one;
3. Send discovery packet to see if we joined any network.

NMK bruteforce $\neq$ good

Bruteforcing the NMK could be long and difficult depending on user's password policy.

Summary

Market researches

First we need an overview of possible DAK passphrase generation.

In the markets

Market researches

First we need an overview of possible DAK passphrase generation.

In the markets

At ebay, leboncoin.fr...

- there people take pictures of every possible positions of the device
- these information could be helpful to study the pattern

Market researches

First we need an overview of possible DAK passphrase generation.

In the markets

At ebay, leboncoin.fr...

- there people take pictures of every possible positions of the device
- these information could be helpful to study the pattern

Found pattern

The DAK passphrase pattern can be represented with this simple regex: `[A-Z]{4}-[A-Z]{4}-[A-Z]{4}-[A-Z]{4}`.

Market researches

First we need an overview of possible DAK passphrase generation.

In the markets

At ebay, leboncoin.fr...

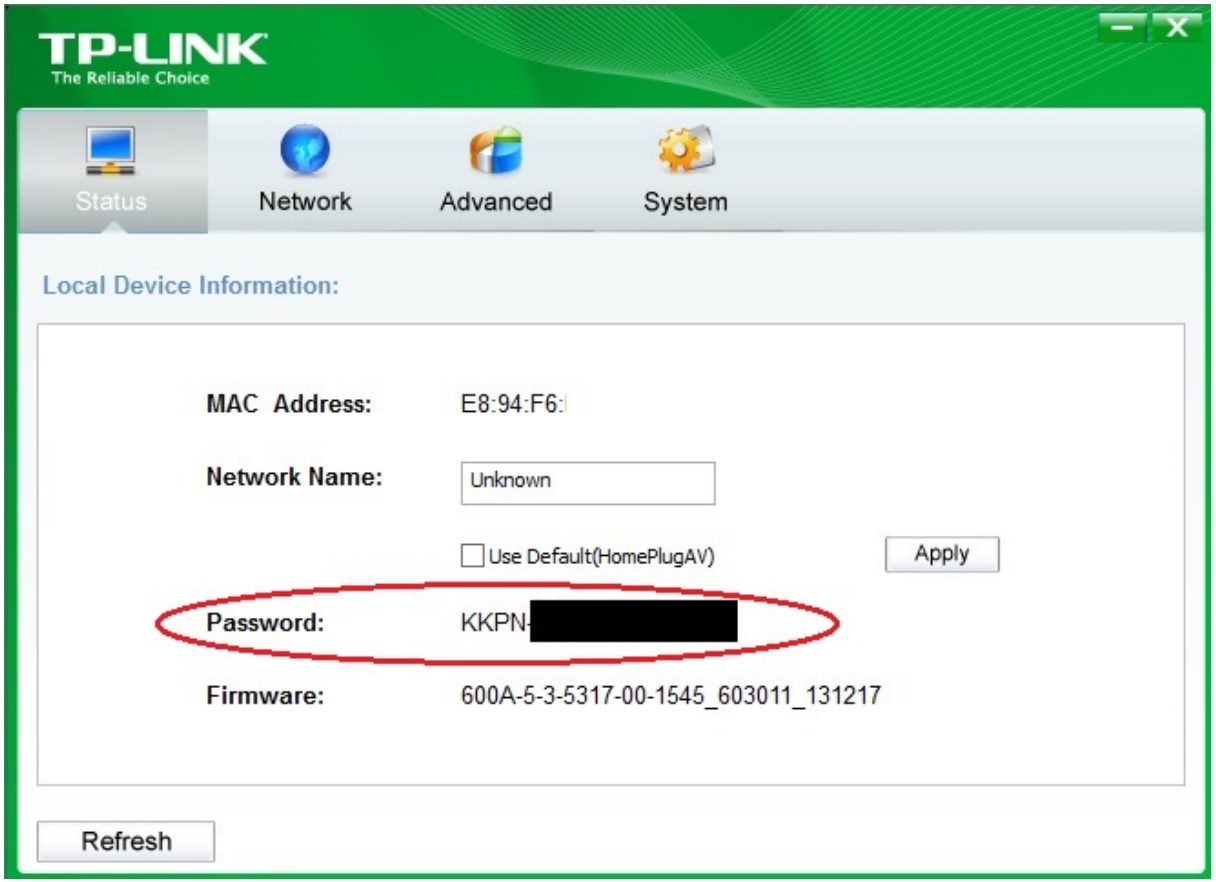
Found pattern

The DAK passphrase pattern can be represented with this simple regex: `[A-Z]{4}-[A-Z]{4}-[A-Z]{4}-[A-Z]{4}`.

Pattern bruteforce

The bruteforce of this pattern is painful! Is there any other way?

TP-Link utility seems to recover DAK passphrases



A little packet analysis... : ReadModuleDataConfirmation

Analysing the packet, the only thing we see are the hash of DAK at offset 0x12 (hidden here), and NMK at offset 0x64 with value=0x50d3e4933f855b7040784df815aa8db7(=HomePlug).

```
>>> hexdump(pkt.ModuleData)
[...]
0020   14 D1 00 00 41 74 68 65  72 6F 73 20 48 6F 6D 65   ....Atheros Home
0030   50 6C 75 67 20 41 56 20  44 65 76 69 63 65 00 00   Plug AV Device..
0040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0060   00 00 00 00 50 D3 E4 93  3F 85 5B 70 40 78 4D F8   ....P...?.[p@xM.
0070   15 AA 8D B7 74 70 76 65  72 5F 36 30 33 30 31 31   ....tpver_603011
0080   5F 31 33 31 32 31 37 5F  30 30 32 00 00 00 00 00   _131217_002.....
```

The question?

How this software can possibly recover this passphrase in a second? Is it derivated from somewhere?

Analysing vendor DLLs

Looking on vendor software we can found a very interesting string %02X%02X%02X%02X%02X%02X (.rdata section) in "PLCOperApi.dll" file.

```
MACProcess:
movzx   ecx, byte ptr [eax+5]
movzx   edx, byte ptr [eax+4]
push    ecx
movzx   ecx, byte ptr [eax+3]
push    edx
movzx   edx, byte ptr [eax+2]
push    ecx
movzx   ecx, byte ptr [eax+1]
push    edx
movzx   edx, byte ptr [eax]
push    ecx
push    edx
lea     eax, [esp+38h+var_14]
push    offset a02x02x02x02x02 ; "%02X%02X%02X%02X%02X%02X"
push    eax             ; char *
call    _sprintf
add     esp, 20h
mov     edx, 6
lea     esp, [esp+0]
```

Good starting point

It's called by "GetLocalDevInfo" that retrieves informations sending a "ReadModuleDataRequest" for PIB, and derives the MAC address to form the DAK key.

Implementation of the DAK generator

Once we have implemented the algorithm, we test it:

```
% python2 genDAK.py f0:de:f1:c0:ff:ee
QFLX-EFRE-QTGC-SZB
% python2 PBKDF1.py QFLX-EFRE-QTGC-SZB
PBKDF1 print: 13a7af2789ddcc19d97075d8efeaf506
```

Then we use the key-derivation function PBKDF1 to output the 16 bytes and send it to the device remotely (we can broadcast it):

```
1  ###[ HomePlugAV  ]###
2      version   = 1.0
3      HPtype    = 'Set Encryption Key Request'
4      OUI       = 0xb052
5      ###[ SetEncryptionKeyRequest ]###
6          EKS       = 0x1
7          NMK       = ''
8          PayloadEncKeySelect= 0x0
9          DestinationMAC= ff:ff:ff:ff:ff:ff
10         DAK       = "\x13\xa7\xaf'\x89\xdd\xcc\x19\xd9pu\xd8\xef\xea\xf5\x06"
```

If the device confirms it → we win!

How powerful is K.O.DAK?

Here is a summary table of bruteforcing techniques difficulties:

Bruteforce technique	Possibilities
DAK passphrase	26^{16}
K.O.DAK classic	256^{6}
K.O.DAK with vendor bytes	256^{3}

Devices with a Qualcomm chip are affected

We have also found a PLC toolkit in github[a], and we can be sure that most of the device could be attacked this way as long as vendors use Qualcomm Atheros DAK passphrase generator.

[a] `https://github.com/qca/open-plc-utils`

Our results

Here is a summary table of possible attacks on different PLCs:

PLC Providers	Ethernet	NMK bruteforce	K.O.DAK Attack
Qualcomm Atheros PLC	YES	YES	YES
INTELLON	YES	YES	MAYBE
ISP PLC	YES	YES	NOT ALL Devices

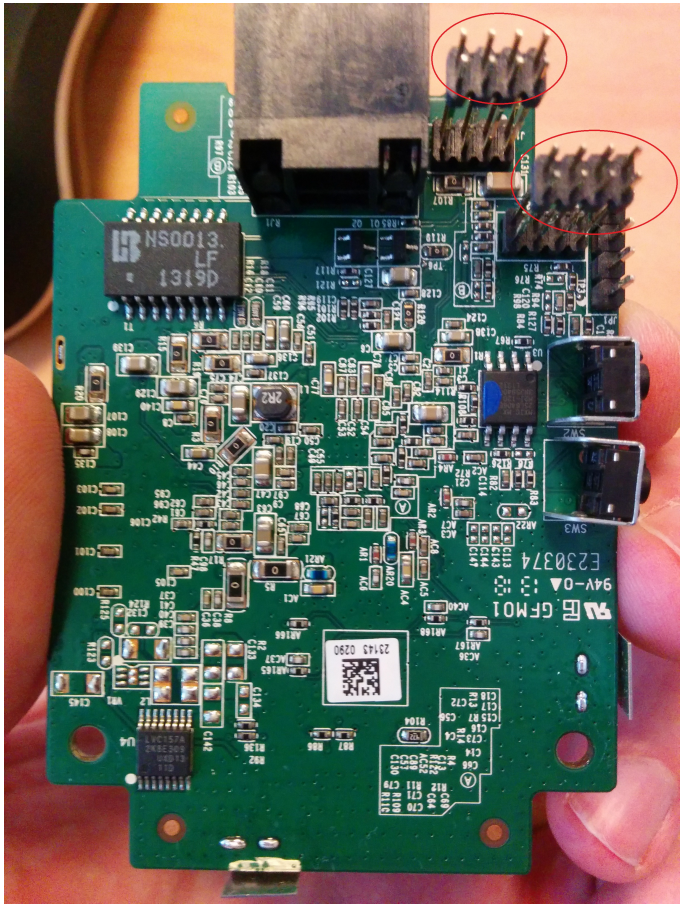
Freeplugs not affected

Freeplugs don't use Qualcomm DAK generator. This is reasuring because Free.fr serves more than 5 702 000 users in France [a], and provides PLCs with their router and STBs for years.

[a] `francois04.free.fr`

Summary

The hardware: remember?

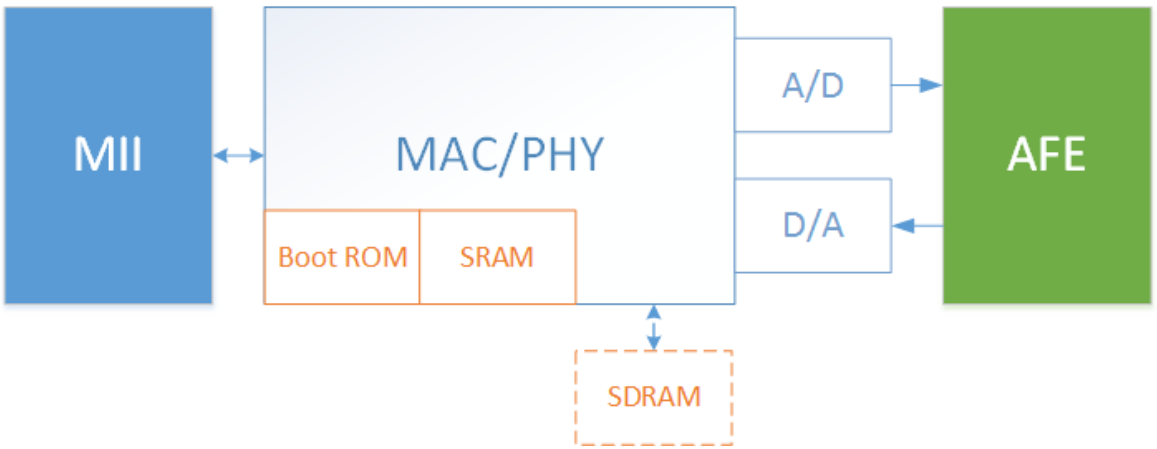


Vendor part

PLC part

The strange ports?

- The two previous ports → MII (Media Independent Interface), or GPSI (General Purpose Serial Interface)
- They connect the PLC MAC/PHY transceiver to IEEE802.3 Ethernet MAC controllers



UART/serial ports could be present on old models, to respond with AT commands[1]

[1] `https://github.com/qca/open-plc-utils/tree/master/serial`

JTAG/serial/UART/... accesses → forget about it!

With the vendor part, we have read/write accesses to the PIB and IMG parts on the NVM !

3 parameters for the "Read Data Module Request"

1. part of the memory : "MAC Soft-Loader Image" (0x0), "MAC Software Image" (0x01), "PIB" (0x02);
2. offset;
3. and the length.

```
###[ HomePlugAV  ]###
    version   = 1.0
    HPtype    = 'Read Module Data Request'
    OUI       = 0xb052
    ###[ ReadModuleData ]###
        ModuleID  = PIB
        reserved  = 0x0
        Length    = 1024
        Offset    = 5120
```

Writing into the memory example

```
###[ HomePlugAV  ]###
    version   = 1.0
    HPtype    = 'Write Module Data Request'
    OUI       = 0xb052
    ###[ WriteModuleData ]###
        ModuleID  = PIB
        reserved  = 0x0
        DataLen   = 1024
        Offset    = 0
        checksum  = 975459083
        ModuleData= '\x05\x07\x00\x008@\x00\x00\xb1\x15)#
        [...]
```

Tip

For the PIB region, you need to overwrite it's PIB checksum32 (at offset 0x8) and send a "WriteModuleDataToNVMRequest" to apply the configuration.

Other cool functionnalities!

The Sniff command that gives details about frame control and beacon.

140	158.140775000	WistronI_b	Broadcast	HomePlug	21	MAC Management, Sniffer Request
141	158.141081000	Tp-LinkT_e	WistronI_b3	HomePlug	60	MAC Management, Sniffer Confirmation
142	158.141474000	Tp-LinkT_e	WistronI_b3	HomePlug	186	MAC Management, Sniffer Indicate
143	158.153746000	Tp-LinkT_e	WistronI_b3	HomePlug	186	MAC Management, Sniffer Indicate
144	158.193671000	Tp-LinkT_e	WistronI_b3	HomePlug	186	MAC Management, Sniffer Indicate
145	158.233831000	Tp-LinkT_e	WistronI_b3	HomePlug	186	MAC Management, Sniffer Indicate
146	158.273699000	Tp-LinkT_e	WistronI_b3	HomePlug	186	MAC Management, Sniffer Indicate
147	158.313759000	Tp-LinkT_e	WistronI_b3	HomePlug	186	MAC Management, Sniffer Indicate

Work in progress

Other commands could be interesting to discover like VS_WRITE_AND_EXECUTE_APPLET or VS_MICROCONTROLLER_DIAG. We will dig a little more to know if we can execute any other applet or try to communicate with the microcontroller.

Gathering CCos MAC address

Enabling the Sniff command we can recover MAC addresses of CCos close to us[2]:

```
###[ SnifferIndicate ]###
    SnifferType= Regular
    Direction = Tx
    SystemTime= 399103809
    BeaconTime= 43033
    ShortNetworkID= 0x80
    [...]
    ###[ Raw ]###
    load = '\x01\xfd40[...]
    [...]
>>> hexdump(pkt.load)
0000    XX XX XX XX XX XX XX XX XX XX XX XX XX XX E8 94     XXXXXXXXXXXXXX..
0010    F6 XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX     .XXXXXXXXXXXXXX
[...]
```

One CCo MAC address is present at address 0xe (begining with bytes: E8 94 F6).

[2]Independently discovered by Ben Tasker: https://www.bentasker.co.uk/documentation/security/282-infiltrating-a-network-via-powerline-homeplugav-adapters

Demos

- Discovery in and out of a AVLN node
- Monitoring and targeting CCos
- Remote CCo configuration to infiltrate a LAN
- Reading target's memory

Archievement

- We have made a scapy Layer that helps us to mess with HomePlugAV protocol (to be completed) and parse the PIB
- This layer can be used to fuzz the client side (vendor's utility)
- HomePlugAV sold in the market are vulnerable to K.O.DAK attack, but not the most used Freeplugs (for the moment)
- If we know the DAK passphrase or we have any access to the device by it's ethernet interface $\rightarrow$ arbitrary read/write access

Work in progress

- Firmware disassembling $\rightarrow$ add other cool functions
  $\Rightarrow$ We could mess with the authentication messages
- Learn more about "applets" that PLC executes

Thank you! ;)

Any questions?