

# EBV & Infineon

## Security Deep Dive – IoT Threats & Countermeasures



prepared by  
Uros Mali, Director Segment Smart Sensing & Control

October 2023



# IoT Security Threats & Countermeasures

## Security Consultants & Hackers Methodology

- Finding open doors into the system
- Understanding the application communication flow
- Disrupting the hardware and reversing the software

## Threats & Countermeasures

- Extracting the software
- Modifying and running the software
- Stealing secrets
- Spoofing the device identity





# Security Consultants & Hackers Methodology



# Security Consultants & Hackers Methodology



Both share a similar gradual strategy to evaluate an IoT solution:

- First they try to identify any easy access to the device through “*open doors*” (Stage 1).
- Then they try to understand the application communication flow (Stage 2).
- Finally they try to disrupt the hardware and/or reverse engineer the software (Stage 3).



Though the evaluation goals are obviously different...

- Security consulting firms are paid to evaluate a solution robustness for a period of time defined by contract, then they typically issue a detailed test report to the customer with security recommendations.
- Hackers stop investigations as soon as device access (being full or limited) can be used to generate revenue. They have NO time limit.



# Finding Open Doors into the IoT Device

- Search online for **known exploits** about device software or hardware.
- Check for **open** or **obvious administrator access (root)**.
- Use **brute force attack** to break weak administrator credentials using **dictionary**.
- Verify if **JTAG** port is still accessible (open).

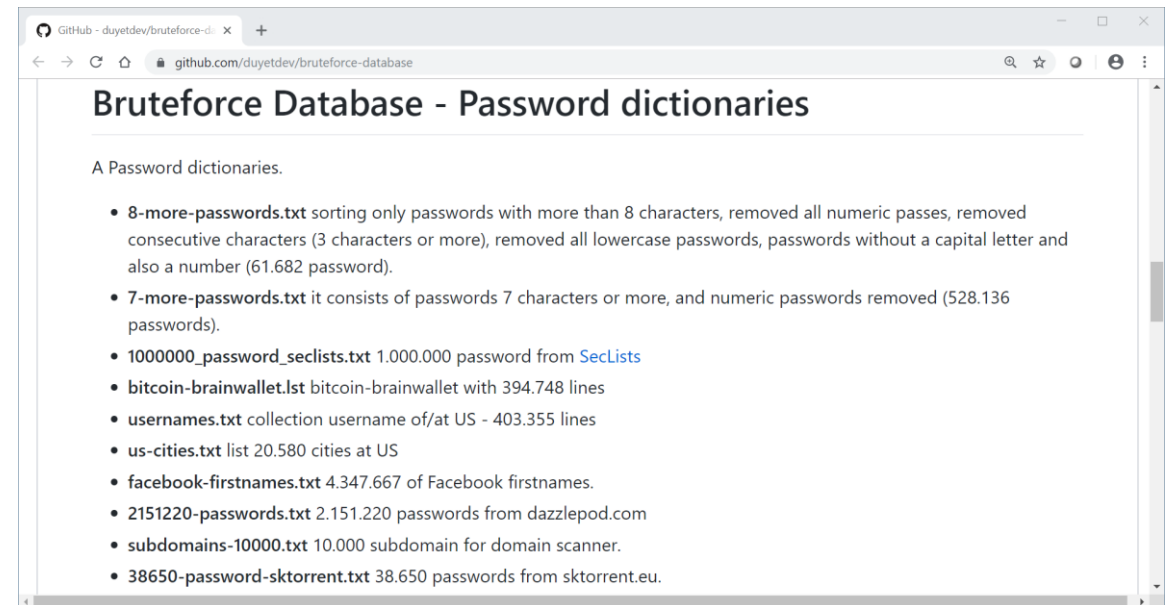
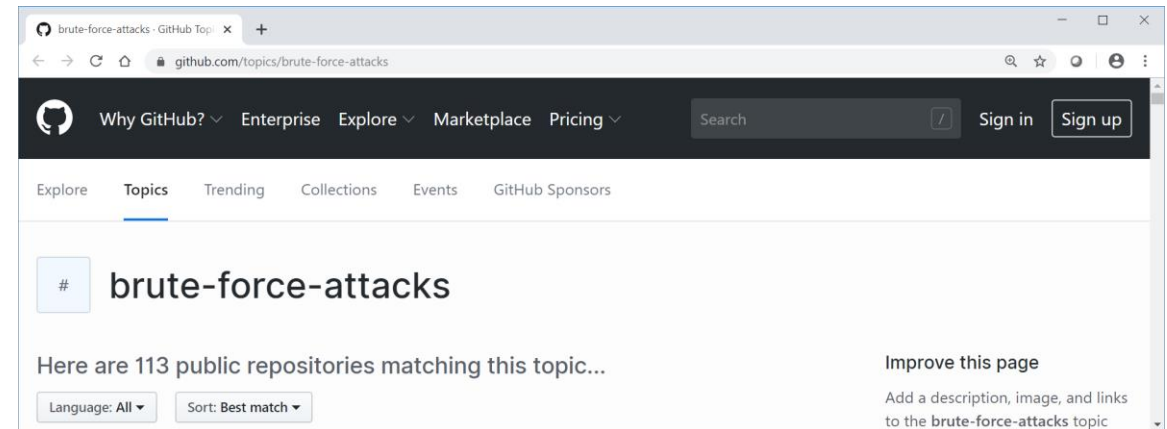


# Stage 1: Brute Force Attack to Get Device Access 1/3

**Brute Force** attack is the simplest method to gain access to a device or server (or anything that is password protected).

It tries various combinations of usernames and passwords again and again until it gets in. **Dictionary** can be used to increase success rate.

Scripting languages such as **Python** are good options to implement brute force attack for embedded targets.





## Stage 1: Trying to Access JTAG 2/3

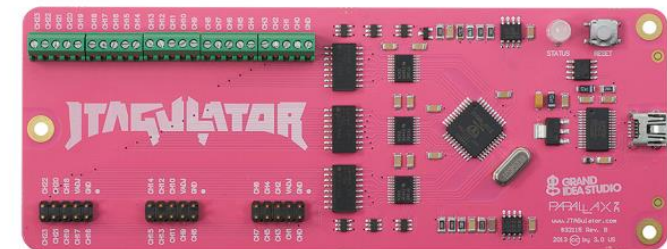
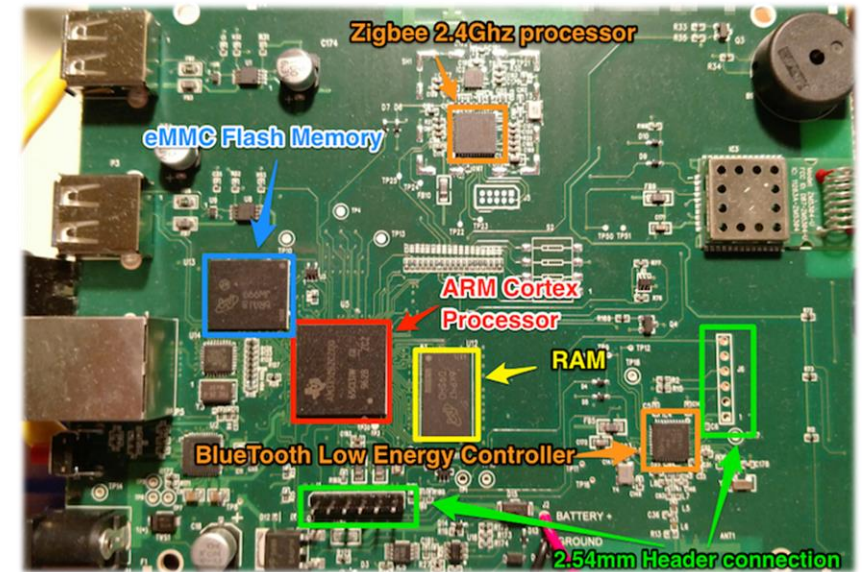
**JTAG** is a common hardware interface that provides your computer with a way to communicate directly with the MCU.

**OpenOCD** is a free software working on top of JTAG, which enables on-chip debugging, in-system programming and testing. It is supported by most modern embedded target.

Inspecting the device PCB can give valuable information:

- MCU reference, memory type, connectivity modules & part numbers.
- PCB revision.
- Headers and test points that might be used as main processor output or programming interface (JTAG).

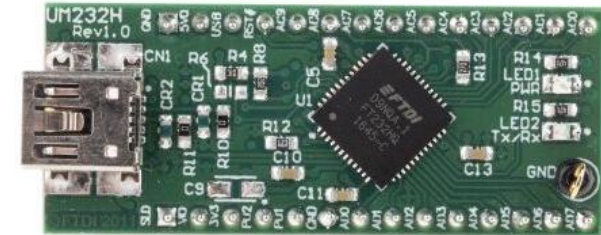
Hardware tool such as *JTAGULATOR* can come in handy for identifying JTAG or UART ports. The tool allows to connect all the pins of a header and automatically walk through and test all possible pinout combinations to identify JTAG or UART.



## Stage 1: Trying to Access JTAG 3/3

Once JTAG pinout is found, any JTAG probe such as *UM232H* can be used to connect the target MCU to a laptop.

From the laptop, run **openocd** and if JTAG port is open, a full access to the device is granted, allowing tools such as **gdb** to access memory and extract firmware:



```

Activities  Terminal ▾ Sat 16:13 ●
hollyw@localhost:~
File Edit View Search Terminal Tabs Help
hollyw@localhost:~ x hollyw@localhost:~ x hollyw@localhost:~ x + ▾
[hollyw@localhost ~]$ openocd -f /usr/share/openocd/scripts/interface/ftdi/um232h.cfg -f /home/hollyw/config2.
cfg
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
adapter speed: 1000 kHz
jtag
Info : clock speed 1000 kHz
Info : JTAG tap: foobar.tap tap/device found: 0x0000100f (mfg: 0x007 (Hitachi), part: 0x0001, ver: 0x0)
Warn : gdb services need one or more targets defined
Info : accepting 'telnet' connection on tcp/4444
  
```

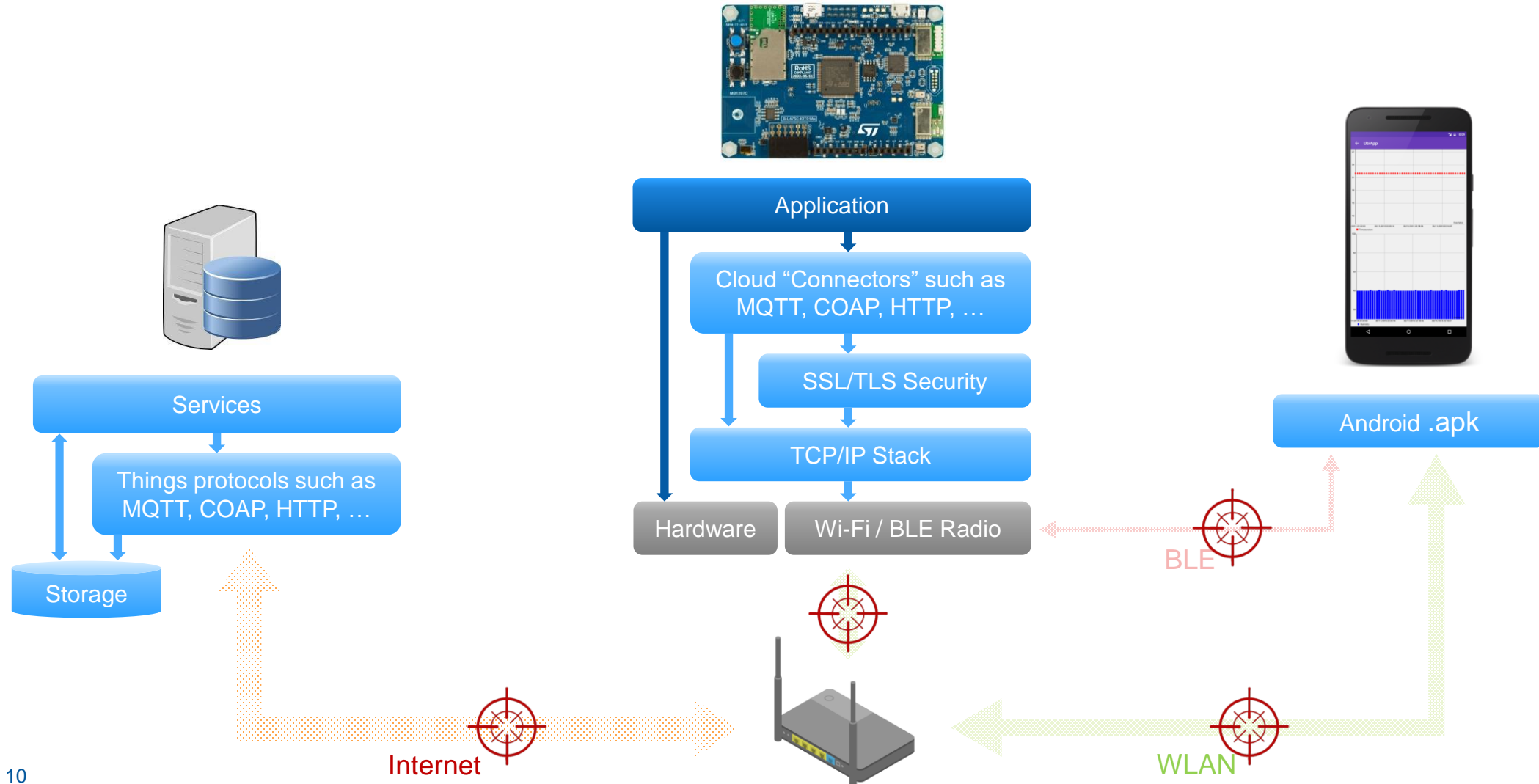


# Understanding the Application Communication Flow

- Search for debug UART interface that could **reveal useful information**.
- **Analyze network** communication.
  - IoT configuration tools such as Android apps are usually not secured and easy to **decompile** and **analyze**.
  - Use **man in the middle attack** with spy router to replay TCP transaction to try to take control of the IoT device.
  - Use **replay attack** probing any data bus, especially between MCU and connectivity module (Wi-Fi, BLE, NB-IoT, etc).



## Stage 2: Analyze Network Communications 1/5



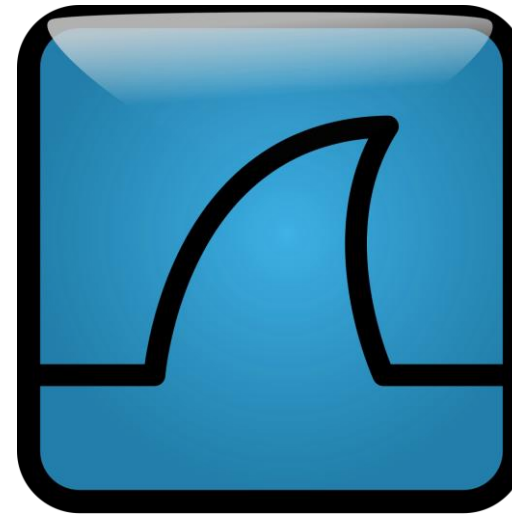


## Stage 2: Analyze Network Communications 2/5



SHODAN

Obtain information about  
a public IP address.



WIRESHARK

Free open source packet  
analyzer.

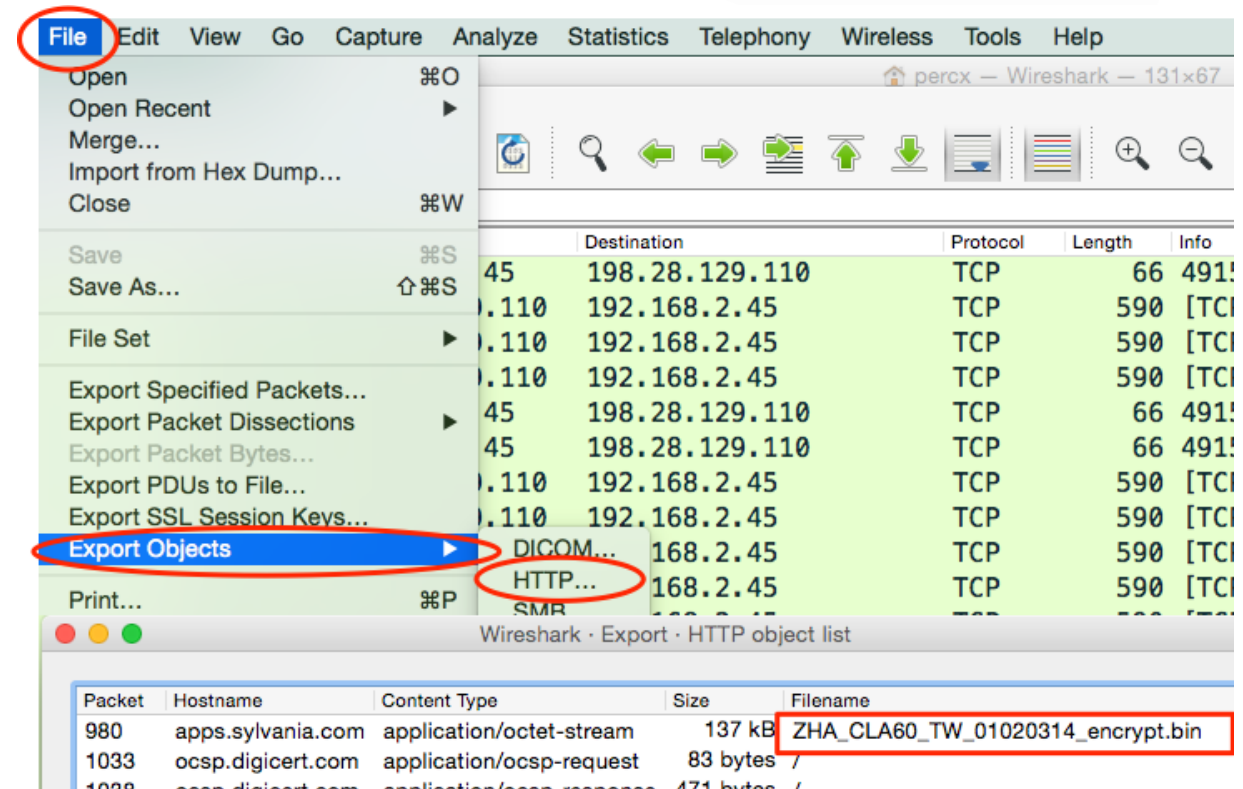
## Stage 2: Analyze Network Communications 3/5

## AirPcap USB-based adapters capture 802.11 wireless traffic for analysis by Wireshark.

**Wireshark** can then be used to listen to the air traffic without modifying the initial network setup. It can also decipher **WPA** (1, 2 or 3) encrypted frames if configured with the right credentials.

Much information can be exposed:

- Overall IP socket topology of the IoT device.
- Protocol being used (plain TCP, TLS, etc).
- Successfully triggering the device OTA process can expose both remote OTA server IP address with file path along with the OTA firmware itself.





## Stage 2: Analyze Android Mobile App 4/5

**Dex2jar** is a set of tools to work with android .dex and java .class files:

- Read/write the Dalvik Executable (.dex) file.
- Convert .dex file to .class files (zipped as jar).
- Disassemble dex to smali files and assemble dex from smali files.



DEX2JAR

**JD-GUI** is a standalone graphical utility that displays Java source codes of .class files. It make it possible to browse the reconstructed source code with the JD-GUI for instant access to methods and fields. It makes it very easy to navigate through the code.



JD-GUI

**ADB Tool** is a development tool that facilitates communication between an Android device and a personal computer.

It is the “Swiss-army knife” of Android development. Most app can give valuable information just by looking at their debug output!

```
Select Command Prompt
07-06 14:38:36.998 601 1517 D NetlinkSocketObserver: NeighborEvent(elapsedM
07-06 14:38:37.028 601 1517 D NetlinkSocketObserver: NeighborEvent(elapsedM
07-06 14:38:38.091 1092 5517 I DeviceScanner: [MDNS] Received response from
07-06 14:38:38.091 1092 5517 I DeviceScanner: [MDNS] notifyDeviceOnline: "be
07-06 14:38:38.093 1092 5517 E Publisher: ProcessDatabaseInternal start
07-06 14:38:38.094 1092 5517 I Publisher: Processing device f9315194dcb7c4b4
07-06 14:38:38.095 1092 5517 I CastMediaRouteProvider: onDevicesPublished wi
07-06 14:38:38.095 1092 5517 I CastMediaRouteProvider: buildRouteDescriptorF
oom, connectionState=Disconnected, volume=0, isIdle=true
07-06 14:38:38.095 1092 5517 I CastMediaRouteProvider: Published 1 routes
07-06 14:38:38.096 1092 5517 I CastMediaRouteProvider: 1 Route: bedroom /Cas
07-06 14:38:39.129 23400 23427 I info : FMX: LogCat: Hello logcat
07-06 14:38:43.377 601 667 D ConnectivityService: updateNetworkScore for N
07-06 14:38:46.389 601 667 D ConnectivityService: updateNetworkScore for N
07-06 14:38:46.854 601 665 D WifiStateMachine: starting scan for "prettyfl
07-06 14:38:49.364 1092 5517 I DeviceScanner: [MDNS] Received response from
07-06 14:38:49.365 1092 5517 I DeviceScanner: [MDNS] notifyDeviceOnline: "be
07-06 14:38:49.375 1092 5517 I DeviceScanner: [MDNS] Received response from
```



ADB Tool

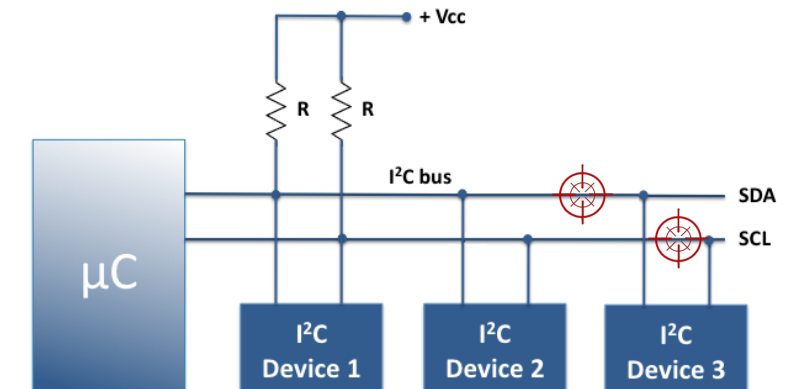
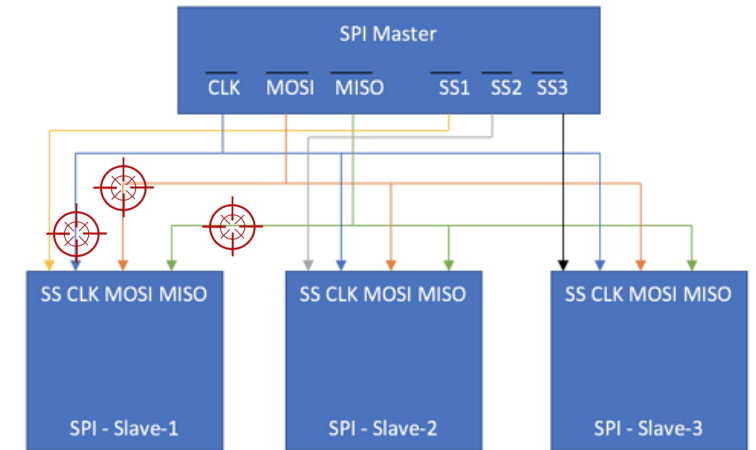
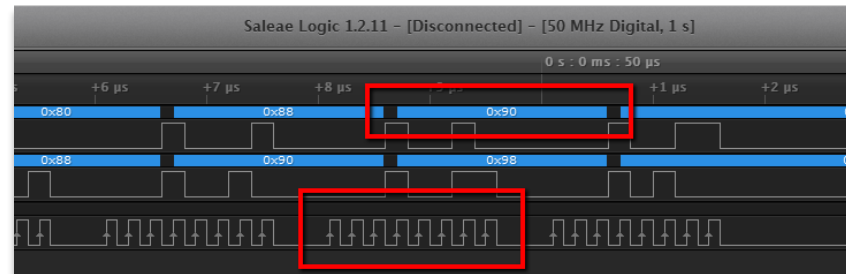
## Stage 2: Probe Data Bus and Replay Attack 5/5

Probing communication data bus is easy to do and the required hardware is quite inexpensive.

**SPI, I2C and UART** data bus are frequently used in embedded device to communicate with external flash memories, connectivity modules, etc...

Some devices are particularly vulnerable to **replay attack** on such data bus:

- AT command based connectivity modules (Wi-Fi, BLE, CATM1) where data is visible in clear.
- Connectivity modules that embed a TLS stack, as data on the bus are very likely to be clear text.
- Any device that does not include bus protection (such counters with encryption) can be easily susceptible to a **replay attack**.





# Disrupting the Hardware and Reversing the Software

- Use **side channel attack** to guess input credentials, bypass firmware instructions or to reveal TLS session keys and expose data communication with remote server.
- Get access to the flash memory content (mostly a matter of time). Use **code injection attack**, relocate external flash to test board to dump memory content or **decapsulate** MCU to read internal flash.
- **Disassemble** firmware and perform **software reversing**:
  - Check for unsafe C library function calls to enable **code injection attack**.
  - Understand boot process to bypass any security when reassembling a **pirate firmware**.
  - Understand firmware update and OTA to enable **large scale attack**.
  - Further **reverse** firmware to alter original program execution flow.
  - **Reveal** weakly **encrypted keys embedded** in the firmware (mostly a matter of time).

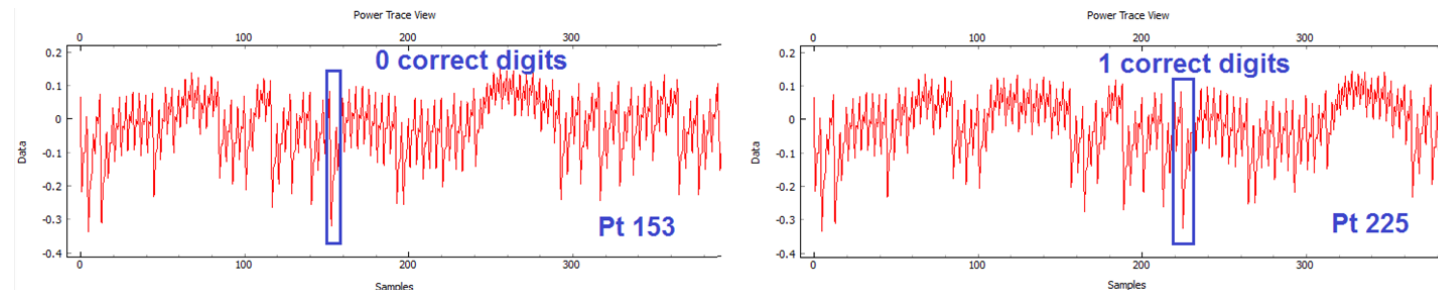


## Stage 3: Timing Attack to Find Credentials with Power Analysis 1/9

**Timing attack** is a **side-channel attack** in which the attacker attempts to compromise a system by analyzing the time taken to execute an algorithm. Every logical operation in a computer takes time to execute, and the time can differ based on the input; with precise measurements of the time for each operation, an attacker can work backwards to the input.

Information can leak from a system through measurement of the time it takes to respond to certain queries. How much this information can help an attacker depends on many variables: cryptographic system design, the CPU running the system, the algorithms used, assorted implementation details, timing attack countermeasures, the accuracy of the timing measurements, etc.

**Timing attacks are often overlooked** in the design phase because they are so dependent on the implementation and can be introduced inadvertently with compiler optimizations. Avoidance of timing attacks involves design of **constant-time functions** and careful testing of the final executable code.



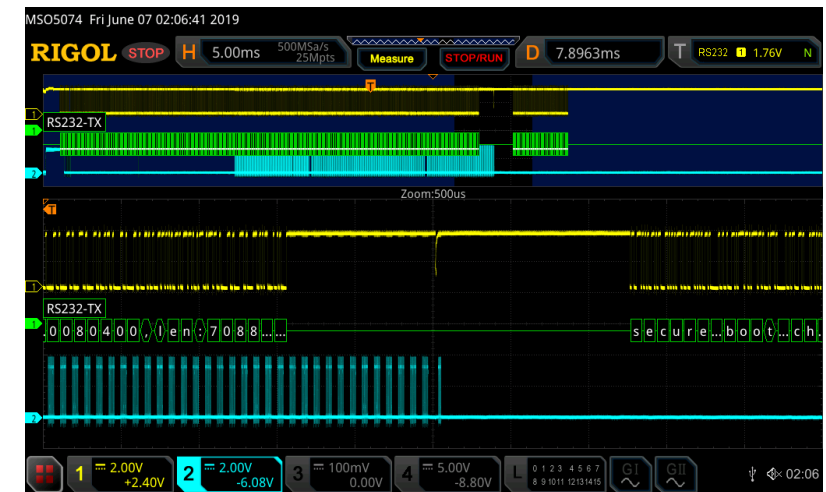
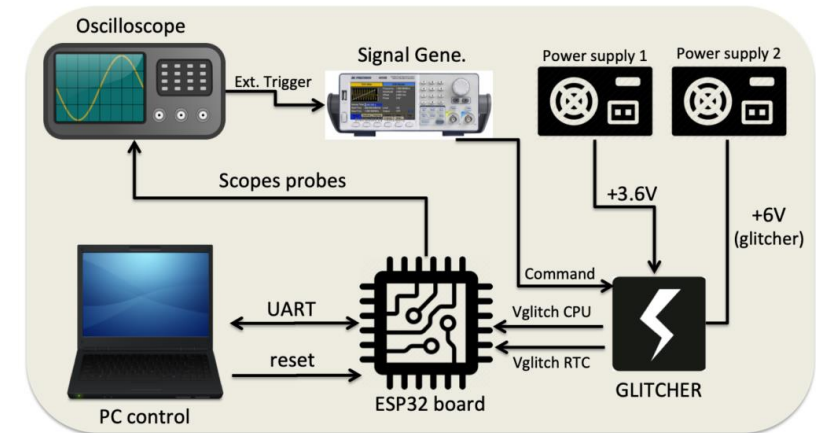
## Stage 3: Glitch Injection to Bypass Instructions in Core Pipeline 2/9

**Glitch** or **Voltage fault injection** is a powerful active side channel attack that modifies the execution-flow of a device by creating disturbances on the power supply line.

The attack typically aims at **skipping security checks** or generating side-channels that gradually **leak sensitive data**, including the firmware code.

The attack typically starts at a **trigger** (for instance fetching the firmware from an external flash), then after an “estimated” delay a voltage glitch is injected, which can bypass some security test instruction in the MCU core pipeline.

A successful attack can take time as it is a matter of try and luck.





## Stage 3: Dumping External Flash Memory 3/9

Going further into hacking an IoT device requires accessing the device firmware. MCU that requires external flash memory are at risk as the flash content can be easily retrieved using tools such as **Flashrom**.



Flashrom

```
C:\Windows\System32\cmd.exe

c:\Users\user\Downloads\_cams.misc\flashrom_CH341A>flashrom -c MX25L12835F/MX25L12845E/MX25L12865E -w 2flash.bin
flashrom v0.9.9-86-gd051d86 on Windows 6.2 (x86)
flashrom is free software, get the source code at https://flashrom.org

Calibrating delay loop... OK.
Found Macronix flash chip "MX25L12835F/MX25L12845E/MX25L12865E" (16384 kB, SPI) on ch341a_spi.
Reading old flash chip contents... done.
Erasing and writing flash chip... Erase/write done.
Verifying flash... VERIFIED.

c:\Users\user\Downloads\_cams.misc\flashrom_CH341A>
```



**Binwalk** is another tool used for analyzing binary files for embedded files and executable code. It is mostly used to extract the content of firmware images.

**Binwalk** can also search for string in the binary files.

```
tools binwalk -e test.bin

DECIMAL      HEXADECIMAL  DESCRIPTION
-----
218040      0x353B8      CRC32 polynomial table, little endian
524288      0x80000      uImage header, header size: 64 bytes, header CRC:
0x4687D1AC, created: 2007-06-15 10:36:26, image size: 2217656 bytes, Data Address:
0x2000000, Entry Point: 0x2000040, data CRC: 0xA54D09E1, OS: Linux, CPU: ARM,
image type: OS Kernel Image, compression type: none, image name: "gm8136"
524352      0x80040      Linux kernel ARM boot executable zImage (little-endian)
542452      0x846F4      gzip compressed data, maximum compression, from Unix, last modified: 1970-01-01 00:00:00 (null date)
3670112     0x380060     xz compressed data
3800908     0x39FF4C     xz compressed data
3931872     0x3BFEE0     xz compressed data
4979008     0x4BF940     xz compressed data
```

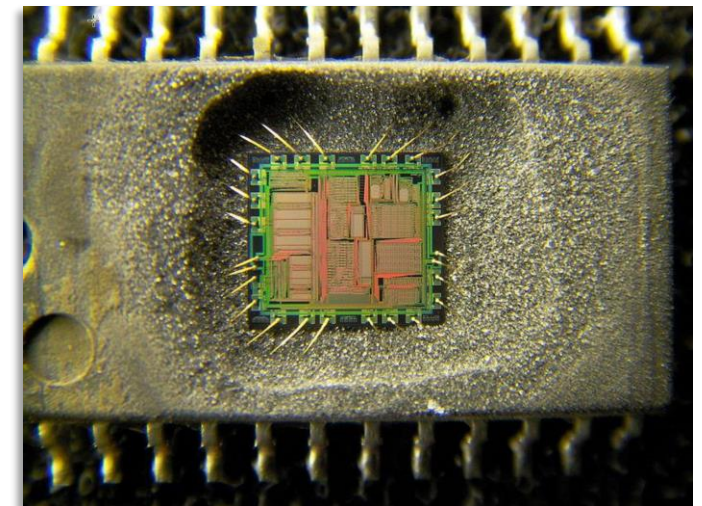
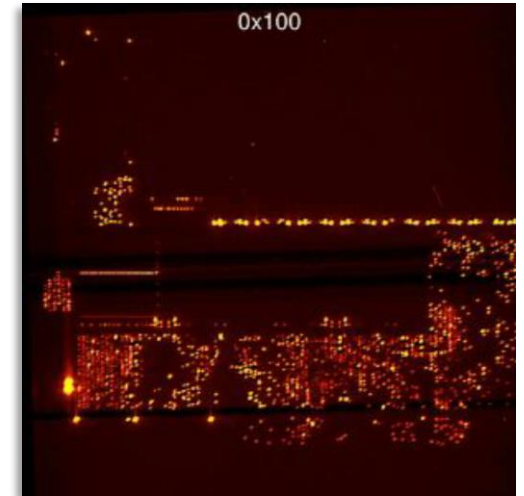
## Stage 3: Dumping Internal Flash Memory (Chip Decapsulation) 4/9

**Decapping** (decapsulation) or **delidding** of an integrated circuit is the process of removing the protective cover of a microchip. This process is typically done in order to debug a manufacturing problem with the chip, or possibly to **extract firmware** from the device.

**Decapping** is usually carried out by chemical etching of the covering, laser cutting, or mechanical removal of the cover.

Plenty of labs offer such services online and the cost of extracting a firmware is not expensive (~2000 USD).

- Semi-invasive attacks:
  - Decapping package.
  - Infrared light/photon emission analysis of backside to find location for attack.
  - Use laser to flip bits (re-enable JTAG) and break crypto.
- Fully-invasive attacks:
  - Much effort but 100% success rate.
  - Modify chip with FIB (Focused Ion Beam).
  - Microprobing.
  - Linear code extraction (LCE).



# Stage 3: Disassemble the Software and Reverse Engineering 5/9

Don't mess with the terminology:

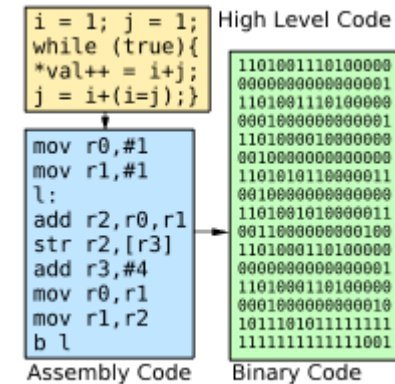
- **Disassemblers** reverse binaries into assembly language.
- **Decompilers** reverse binaries into higher-level languages, like C or C++.
- **Debuggers** allow to view and change the state of a running program.
- **Hex Editors** allow to view and edit the contents of a binary.

Binary reversing tools:

- IDA Pro.
- Radare2.
- Binary Ninja.

Firmware analysis tools:

- **firmwalker** (with **binwalk**, **cpu\_rec**).
- **firmware-analysis-toolkit**.
- **FACT** (Firmware Analysis and Comparison Tool).



IDA Pro



Radare2



Binary Ninja



# Stage 3: Understand Software Flow & Extract Embedded Keys 6/9

IDA - /rootfs/bin/cms.idb (cms)

**Call BSP\_NET\_GetBaseMacAddress()**

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

- ATP\_WLAN\_GetVar
- ATP\_WLAN\_GetMsgProc
- sub\_478344
- sub\_4785DC
- ATP\_WLAN\_WpsSetMsgProc
- ATP\_WLAN\_SetMsgProc
- ATP\_WLAN\_Init** **Function Name**
- sub\_47AC10
- ATP\_XDSL\_Init
- ATP\_XDSL\_UnInit

Line 939 of 1620

Graph overview

Location inside the function

```

jair $t9, BSP_NET_GetBaseMacAddress
addiu $a0, $sp, 0x2A8+var_1E8
lw $gp, 0x2A8+var_280($sp)
lbu $a2, 0x2A8+var_1E4+1($sp)
lbu $v1, 0x2A8+var_1E8+2($sp)
lbu $a0, 0x2A8+var_1E8+3($sp)
lbu $a1, 0x2A8+var_1E4($sp)
lbu $v0, 0x2A8+var_1E8+1($sp)
sw $a2, 0x2A8+var_288($sp)
la $a2, aUleDDevNull # "uletd >/dev/null"
la $t9, snprintf
lbu $a3, 0x2A8+var_1E8($sp)
sw $v1, 0x2A8+var_294($sp)
sw $a0, 0x2A8+var_290($sp)
sw $a1, 0x2A8+var_28C($sp)
addiu $a2, (a02x02x02x02x_0 - 0x4A0000) # "%02x%02x%02x%02x%02x"
move $a0, $a1
li $a1, 0x10
jalr $t9, snprintf
sw $v0, 0x2A8+var_298($sp)
lw $gp, 0x2A8+var_280($sp)
move $a0, $a1
la $a1, aUleDDevNull # "uletd >/dev/null"
la $t9, memcmp
addiu $a1, (a001020304050 - 0x4A0000) # "001020304050"
jalr $t9, memcmp
li $a2, 0xC
lw $gp, 0x2A8+var_280($sp)
lbu $a1, 0x2A8+var_1E4($sp)
lbu $v1, 0x2A8+var_1E4+1($sp)
la $a0, g_astWlVar
la $a2, aUleDDevNull # "uletd >/dev/null"
lbu $a3, 0x2A8+var_1E8+3($sp)
la $t9, snprintf
andi $v0, $a1, 0xFF
andi $v1, 0xFF
sw $v1, 0x2A8+var_294($sp)
addiu $a0, (unk_508C10 - 0x508C04)
$ a2, (aTalktalk02x02x - 0x4A0000) # "TALKTALK-%02x%02x%02x"
andi $a3, 0xFF
li $a1, 0x21 # '1'
jalr $t9, snprintf
sw $v0, 0x2A8+var_298($sp)
lw $gp, 0x2A8+var_280($sp)
lui $a1, 0xE880
la $a2, g_astWlVar # Segfaults
la $t9, ATP_DBSetPara # segfaults
move $a0, $a3
li $a1, 0xE8801E09
jalr $t9, ATP_DBSetPara
addiu $a2, (unk_508C10 - 0x508C04)
lw $gp, 0x2A8+var_280($sp)
bnez $v0, loc_479C8C
  
```

Perform some operations with the MAC

Build the TALKTALK-\* string with snprintf()

Call ATP\_DBSetPara()

125.00% (5252,5164) (1086,782) UNKNOWN 00479DD0: ATP\_WLAN\_In (Synchronized with Hex Vi

```

loc_404DD0:
la $a0, loc_430000
nop
addiu $a0, (aRouteAddNet239 - 0x430000) # "route add -net 239.0.0.0 netmask 255.0.0.0"
la $t9, system
jair $t9, system
nop
lw $gp, 0x68+var_58($fp)
addiu $v0, $fp, 0x68+var_28
move $a0, $v0
move $a1, $zero
li $a2, 0x14
la $t9, memset
nop
jalr $t9, memset
lw $gp, 0x68+var_58($fp)
nop
la $a0, loc_430000
nop
addiu $a0, (aEtcServercert_ - 0x430000) # "/etc/servercert.pem"
la $t9, strdup
nop
jalr $t9, strdup
nop
lw $gp, 0x68+var_58($fp)
sw $v0, 0x68+var_20($fp)
la $a0, loc_430000
nop
addiu $a0, (aEtcServerkey_p - 0x430000) # "/etc/serverkey.pem"
la $t9, strdup
nop
jalr $t9, strdup
nop
lw $gp, 0x68+var_58($fp)
sw $v0, 0x68+var_1C($fp)
la $a0, loc_430000
nop
addiu $a0, (aEtcRoot_pem - 0x430000) # "/etc/root.pem"
la $t9, strdup
nop
jalr $t9, strdup
nop
lw $gp, 0x68+var_58($fp)
sw $v0, 0x68+var_24($fp)
li $v0, 1
sb $v0, 0x68+var_28($fp)
li $v0, 1
sb $v0, 0x68+var_27($fp)
li $v0, 0x1151
sh $v0, 0x68+var_18($fp)
addiu $v0, $fp, 0x68+var_28
move $a0, $v0
la $t9, ATP_TR064_ConfigSSL
nop
jalr $t9, ATP_TR064_ConfigSSL
nop
lw $gp, 0x68+var_58($fp)
sw $v0, 0x68+var_50($fp)
lw $v0, 0x68+var_50($fp)
nop
beqz $v0, loc_404EF8
nop
  
```

Private RSA key being loaded

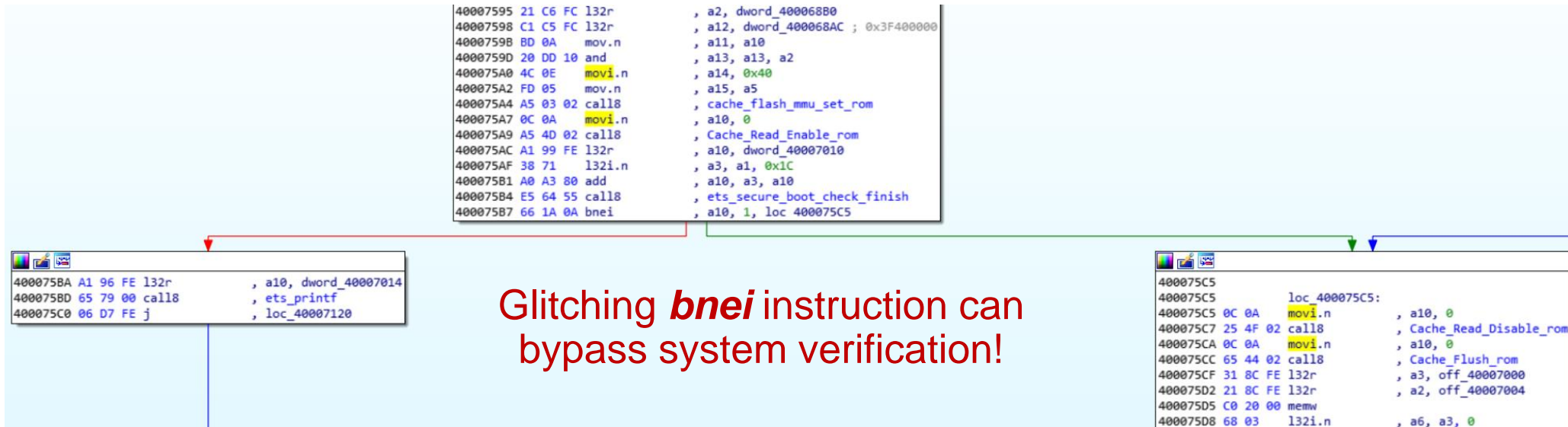
Call to ATP\_TR064\_ConfigSSL

## Stage 3: Bypassing Security Verification (Secure Boot) 7/9

**Secure Boot** is a technology where the system firmware checks that the system boot loader is signed with a cryptographic key, thus authenticating the system boot loader.

Disassembling the **boot ROM** code reveals that the function **ets\_secure\_boot\_check\_finish** performs the crypto verification and then **bnei** instruction (0x400075B7) performs the branch to jump on the firmware image or reset the system.

```
ets Jun  8 2016 00:22:57
rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8708
load:0x40078000,len:17352
load:0x40080400,len:6680
secure boot check fail
ets_main.c 371
...(infinite loop)
```



Glitching **bnei** instruction can  
bypass system verification!

## Stage 3: Code Injection for Potential Large-Scale Attacks 8/9

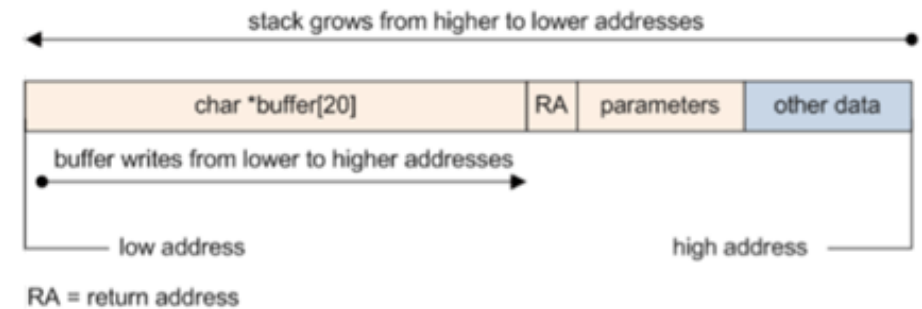
Code injection technique such as **buffer overflow** (or **stack smashing**) rely on weak software implementation where a stack frame gets corrupted by overflowing an input buffer during a function call. Vulnerable implementations typically don't perform boundary check on the input data.

In such case, overflowing the input buffer can overwrite the stack frame of the current function (with its return address) thus allowing for **malicious code injection**.

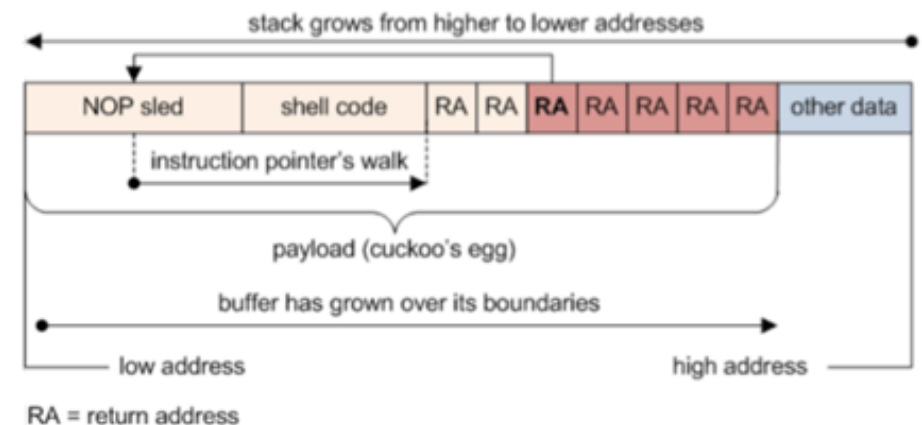
Buffer overflow attacks are typically used to inject **shell code** to get a privileged access (root) in the device.

This exploit can be used in many ways to inject code to negatively affect device quality of service, compromise sensitive user information, turn the device into a botnet, etc.

*Stack before overflow*



*Stack after overflow attack*





## Stage 3: Extract AES Keys from Memory Coredump 9/9

Free Tools such as **findaes** can identify and extract AES strings from a **coredump** file.

Searches for AES keys by searching for their **key schedules**. It can find 128-, 192-, and 256-bit keys, such as those used by TrueCrypt, BitLocker or TLS stacks. Originally intended for memory images but also work with any arbitrary data.

Download link:

<https://sourceforge.net/projects/findaes>

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

c:\Work\Tools\findaes-1.2>findaes.exe teraterm8.log
Searching teraterm8.log
Found AES-256 key schedule at offset 0x29927:
95 a8 08 c8 c2 df 9d 20 3a b0 65 5f 18 c8 60 45 bf dd 16 8b 77 1e 52 8c 14 85 70 f6 3a 8b 09 15
Found AES-128 key schedule at offset 0x30b6b:
3b c8 cc cd 79 d5 76 1e b4 d9 98 33 f2 9e e9 3b

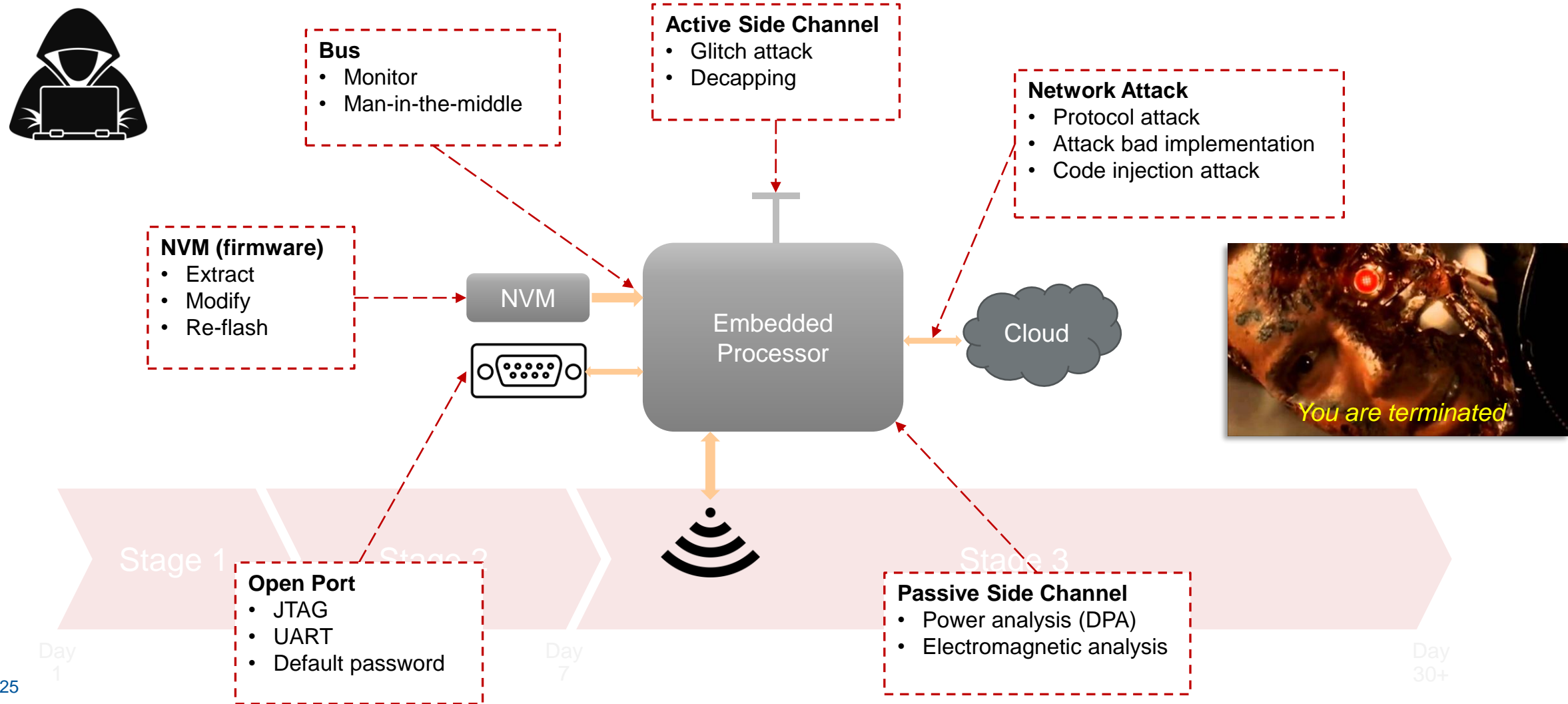
c:\Work\Tools\findaes-1.2>
```

```
/cygdrive/c/Work/Tools/findaes-1.2
RichardT@FRANTY2NB101R97 /cygdrive/c/Work/Tools/findaes-1.2
$ hexkey=3bc8cccd79d5761eb4d99833f29ee93b

RichardT@FRANTY2NB101R97 /cygdrive/c/Work/Tools/findaes-1.2
$ hexiv=169b1aca9fbf2e32c98d28ecbb8f2f88

RichardT@FRANTY2NB101R97 /cygdrive/c/Work/Tools/findaes-1.2
$ ./xxd.exe -r -p report_msg.txt | openssl enc -d -nopad -aes-128-cbc -K $hexkey -iv $hexiv | hexdump -C
00000000 32 c9 03 00 4c 64 65 76 69 63 65 73 2f 33 46 42 |2...ldevices/3FB
00000010 46 44 30 37 46 33 43 38 39 34 39 36 34 41 37 36 |FD07F3C894964A76
00000020 45 34 45 45 44 39 41 37 32 43 34 36 41 2d 30 31 |E4EED9A72C46A-01
00000030 32 33 46 37 36 31 44 44 42 39 36 43 33 46 30 31 |23F761DDB96C3F01
00000040 2f 6d 65 73 73 61 67 65 73 2f 65 76 65 6e 74 73 |/messages/events
00000050 2f 00 04 7b 22 63 70 69 64 22 3a 22 33 46 42 46 |/..{"cpid":"3F8F
00000060 44 30 37 46 33 43 38 39 34 39 36 34 41 37 36 45 |D07F3C894964A76E
00000070 34 45 45 44 39 41 37 32 43 34 36 41 22 2c 22 64 |4EED9A72C46A","d
00000080 74 67 22 3a 22 33 44 44 32 34 43 33 42 2d 37 44 |tg":"3DD24C3B-7D
00000090 45 41 2d 34 38 30 39 2d 42 32 32 33 2d 35 45 30 |EA-4809-B223-5E0
000000a0 38 45 41 38 45 45 38 33 44 22 2c 22 6d 74 22 3a |8EA8EE83D","mt":
000000b0 30 2c 22 73 64 6b 22 3a 7b 22 6c 22 3a 22 4d 5f |0,"sdk":{"l":"M
000000c0 43 22 2c 22 76 22 3a 22 32 2e 30 22 2c 22 65 22 |c","v":"2.0","e"
000000d0 3a 22 41 56 4e 45 54 22 7d 2c 22 64 22 3a 5b 7b |: "AVNET"},"d":[{"
000000e0 22 69 64 22 3a 22 30 31 32 33 46 37 36 31 44 44 |"id":"0123F761DD
000000f0 42 39 36 43 33 46 30 31 22 2c 22 74 67 22 3a 22 |B96C3F01","tg":
00000100 22 2c 22 64 22 3a 5b 7b 22 61 63 63 22 3a 7b 22 |","d":[{"acc":{
00000110 58 22 3a 37 2c 22 59 22 3a 32 33 2c 22 5a 22 3a |x":7,"y":23,"z":
00000120 31 30 31 36 7d 2c 22 67 79 72 6f 22 3a 7b 22 58 |1016},"gyro":{"x
00000130 22 3a 31 34 30 2c 22 59 22 3a 33 35 30 2c 22 5a |":140,"y":350,"z
00000140 22 3a 2d 32 38 30 7d 2c 22 6d 61 67 22 3a 7b 22 |":-280},"mag":{"
00000150 58 22 3a 2d 33 32 34 2c 22 59 22 3a 32 33 31 2c |x":-324,"y":231,
00000160 22 5a 22 3a 2d 34 31 31 7d 2c 22 70 72 65 73 73 |"z":-411},"press
00000170 22 3a 39 39 37 2c 22 68 75 6d 22 3a 34 34 2c 22 |":997,"hum":44,"
00000180 74 65 6d 70 22 3a 32 33 7d 5d 2c 22 64 74 22 3a |temp":23},"dt":
00000190 22 32 30 32 32 2d 31 31 2d 30 39 54 31 36 3a 31 |"2022-11-09T16:1
000001a0 31 3a 30 34 2e 30 30 30 5a 22 7d 5d 2c 22 74 22 |[1:04.000Z"}], "t"
000001b0 3a 22 32 30 32 32 2d 31 31 2d 30 39 54 31 36 3a |: "2022-11-09T16:
000001c0 31 31 3a 30 34 2e 30 30 30 5a 22 7d c9 13 d9 d9 |[11:04.000Z"}....
000001d0 81 b2 4a 01 46 3f ef c3 d8 84 c5 59 07 be 3c b8 |...J.F?....Y...<
000001e0 75 b3 79 d3 30 1c 50 1f 1b 3f 8d 2c 03 03 03 03 |u.y.0.P...?.....
000001f0
```

# IoT Devices Need to Address Multiple Threats!

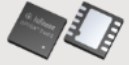



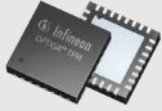




# Security Countermeasures





# OPTIGA™ Family – Embedded Security Solutions

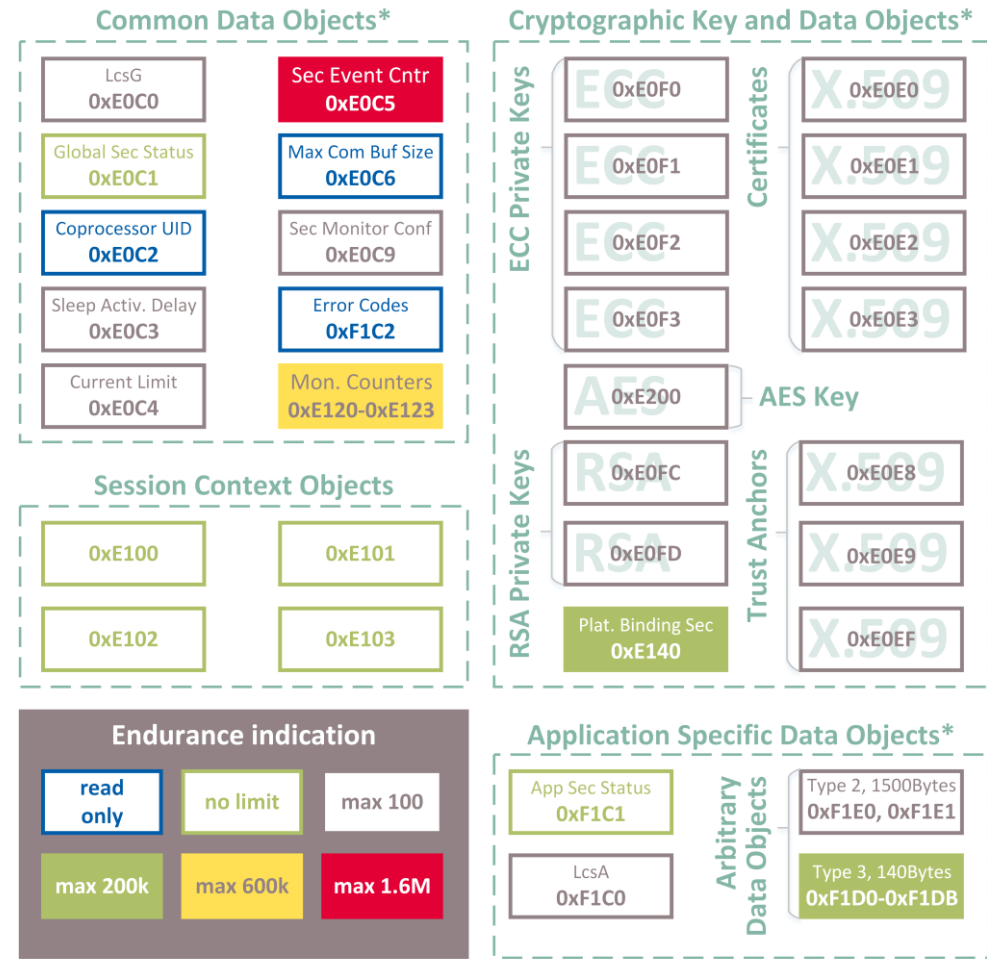
	OPTIGA™ Trust B	OPTIGA™ Trust M	OPTIGA™ Trust M v3	OPTIGA™ Trust X	OPTIGA™ TPM SLB	OPTIGA™ TPM SLM/SLI	OPTIGA™ Connect
							
Security Level	Basic	CC EAL 6+*	CC EAL 6+*	CC EAL 6+*	CC EAL 4+	CC EAL 4+	CC EAL 5+
Functionality	Authentication	Connected device security - Toolbox based	Enhanced Trust M feature-set	Connected device security	TCG standard	TCG standard Automotive/industrial qualified	eSIM
NVM (Data)	64 Byte	10 kByte	10 kByte	10 kByte	6 kByte	6 kByte	Multiple MNO-profiles
Cryptography Private key stored in secure HW	ECC131	ECC384 RSA2K	ECC521 RSA2K	ECC384	ECC256 RSA2K	ECC256 RSA2K	ECC, RSA
Type of Host System	MCU without OS / proprietary OS / RTOS			Embedded Linux		Windows / Linux	
Interface	SWI	I2C	I2C	I2C	I2C, SPI, LPC	SPI	ISO7816
System integration	✓	✓	✓	✓	Platform vendor	Platform vendor	✓

\* Based on certified HW

# OPTIGA™ Trust M - SLS 32AIA010MH/S/K/L

## Key Features

- High-end security controller
- Common Criteria Certified EAL6+ (high) hardware
- Turnkey solution
- Up to 10kB user memory
- PG-USON-10-2,-4 package (3 x 3 mm)
- Standard & Extended temperature ranges
- I2C interface with Shielded Connection (encrypted communication)
- Cryptographic support:
  - ECC : NIST curves up to P-521, Brainpool r1 curve up to 512,
  - RSA® up to 2048,
  - AES key up to 256 , HMAC up to SHA512,
  - TLS v1.2 PRF and HKDF up to SHA512
- Crypto ToolBox commands for SHA-256, ECC and RSA® Feature, AES, HMAC and Key derivation
- Configurable device security monitor, 4 Monotonic up counters
- Protected(integrity and confidentiality) update of data, key and metadata objects
- Hibernate for zero power consumption
- Lifetime for Industrial Automation and Infrastructure is 20 years and 15 years for other Application Profiles



\* The content is either preconfigured by Infineon or is part of the personalisazation offer

OPTIGA™ Trust M Software Framework on Github - <https://github.com/Infineon/optiga-trust-m>

# OPTIGA™ Security Functions – Use Cases



## Device Authentication

- › One-way authentication
- › Mutual authentication



## System integrity

- › Secure Boot
- › Remote platform verification



## Secure Channel

- › Encrypted Communication
- › Key Generation



## Lifecycle Management

- › Key Backup and refurbishment
- › Personalization and identities
- › Supply chain tracking



## User Management

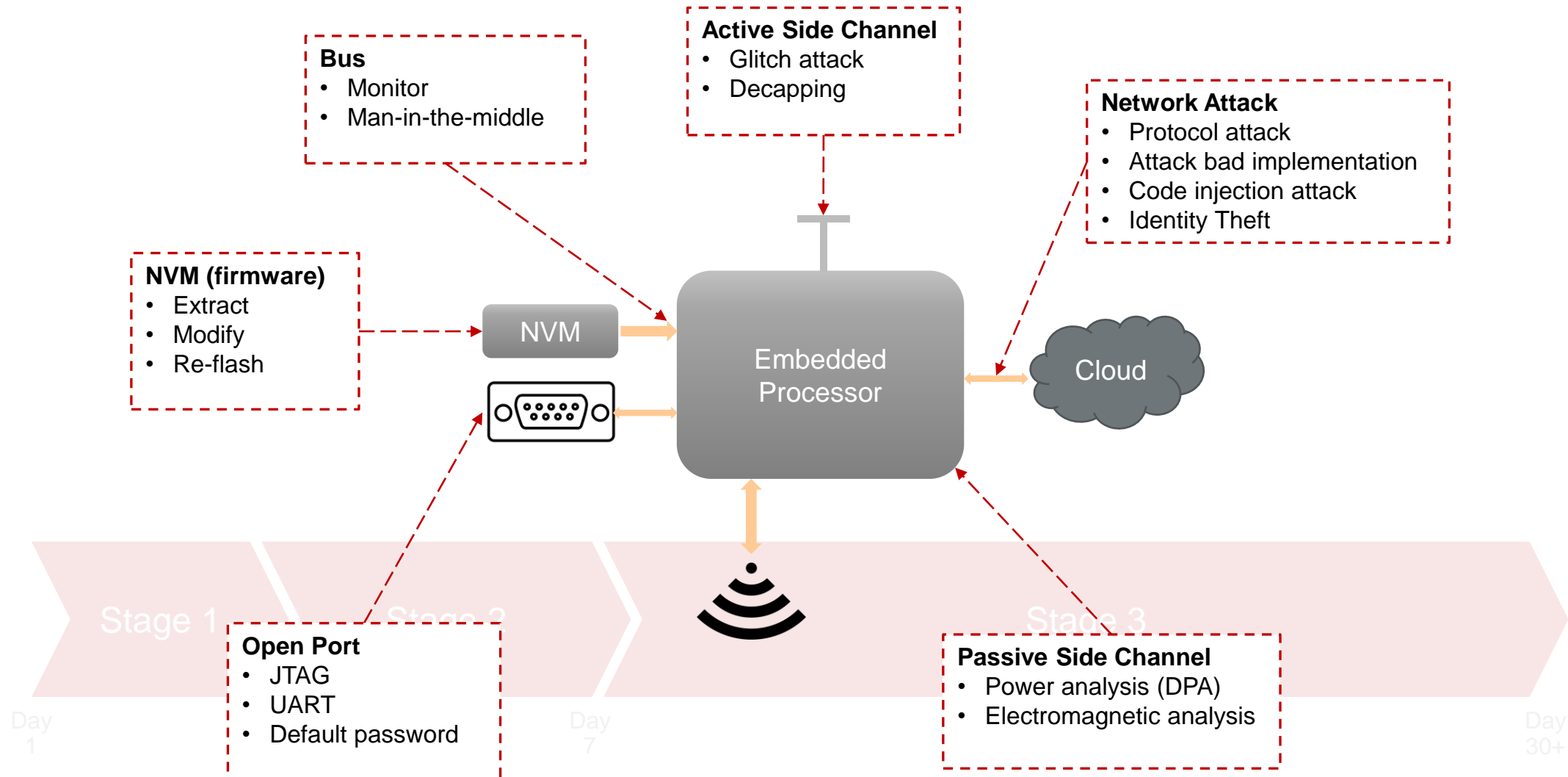
- › Password Protection
- › User management and keys



## Secure Updates

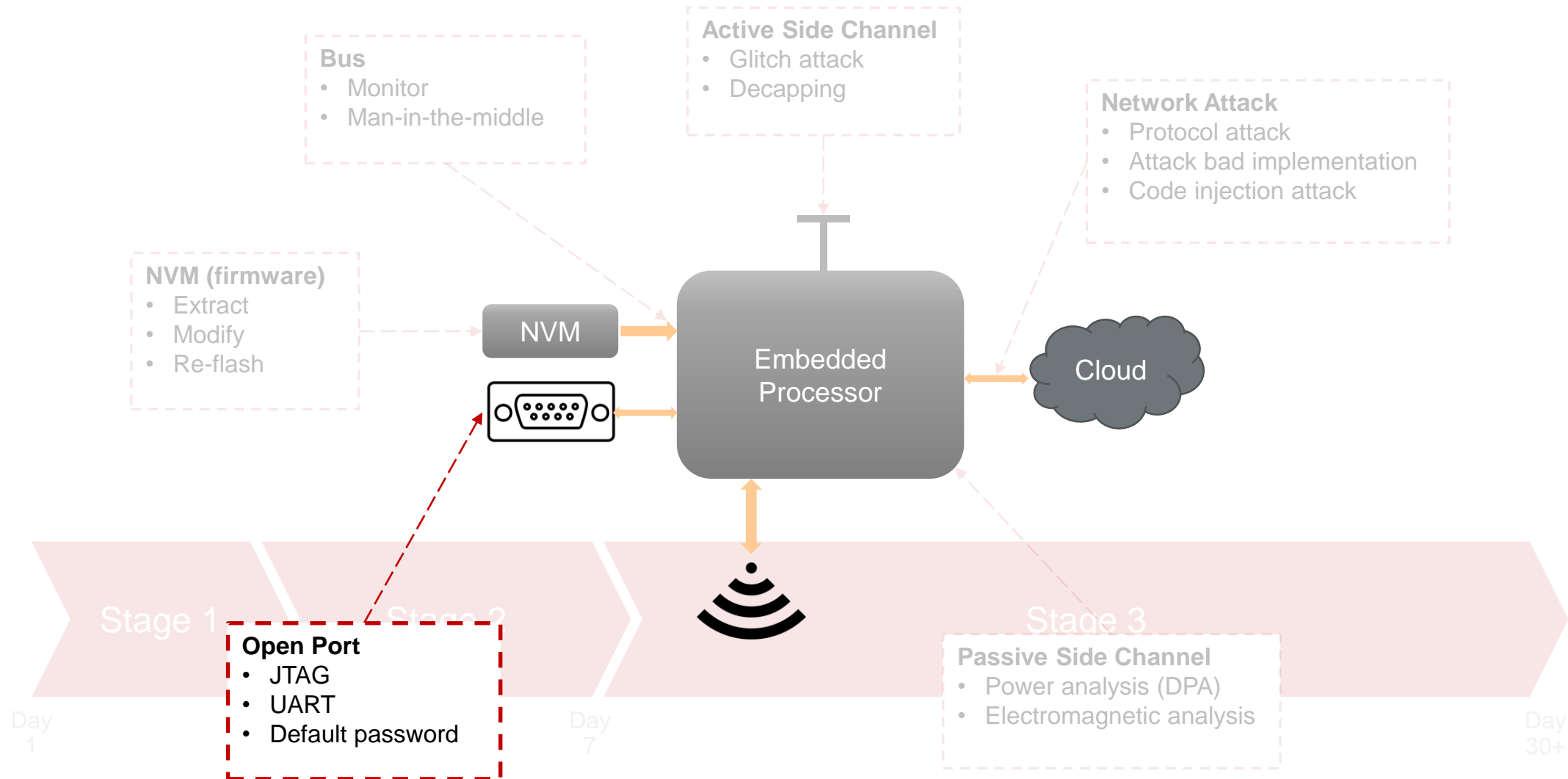
- › Remote maintenance
- › In-field flexibility and reaction

# IoT Devices Need to Address Multiple Threats!





# Ensure Device has NO Open Access



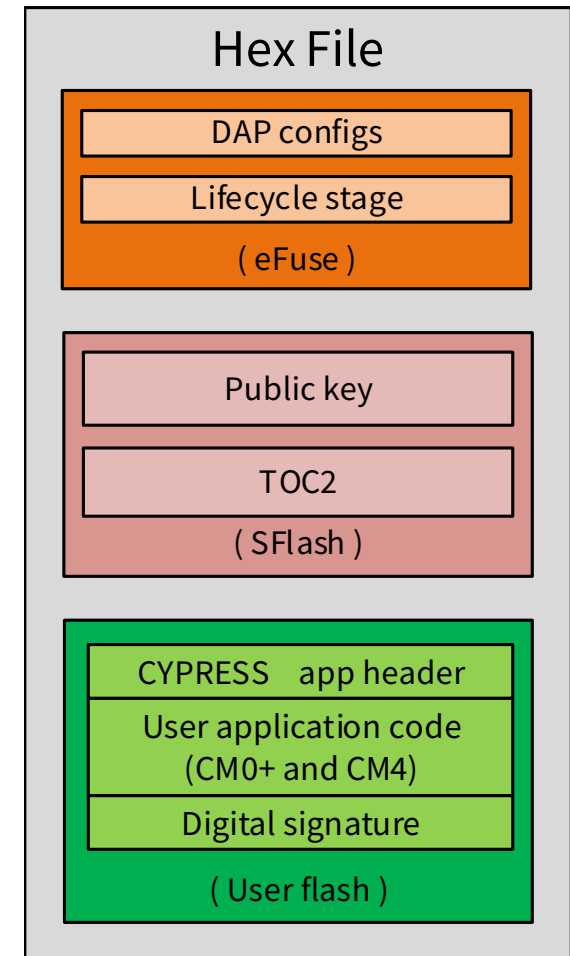
## Use PSoC™ Security Features – Debug Access Port - DAP

There are **three** different access port restriction settings:

- Secure access restrictions (**SAR**)
- Normal access restrictions (**NAR**)
- Dead access restrictions (**DAR**)

The **SAR** and **DAR** are stored in the **eFuse** which cannot be erased once set. These restrictions must be set before entering the **SECURE** lifecycle stage. Once in SECURE lifecycle stage, the SAR and DAR cannot be altered.

The **NARs** are stored in the **SFlash** and are protected only by the system call firmware which ALWAYS runs with a protection context equal 0. The system call functions allow you to **increase** the NAR security, but not to reduce it.

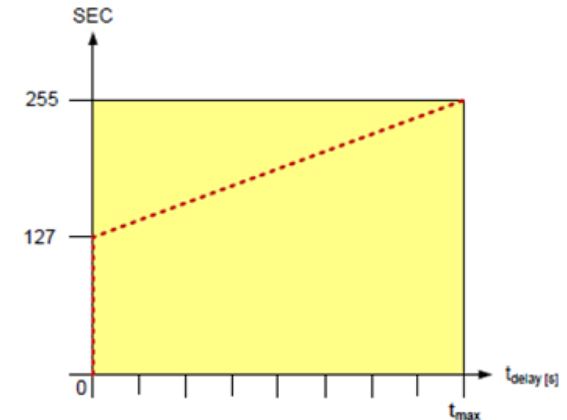


# OPTIGA™ Trust M – Security Monitor

## Security event counter (SEC)

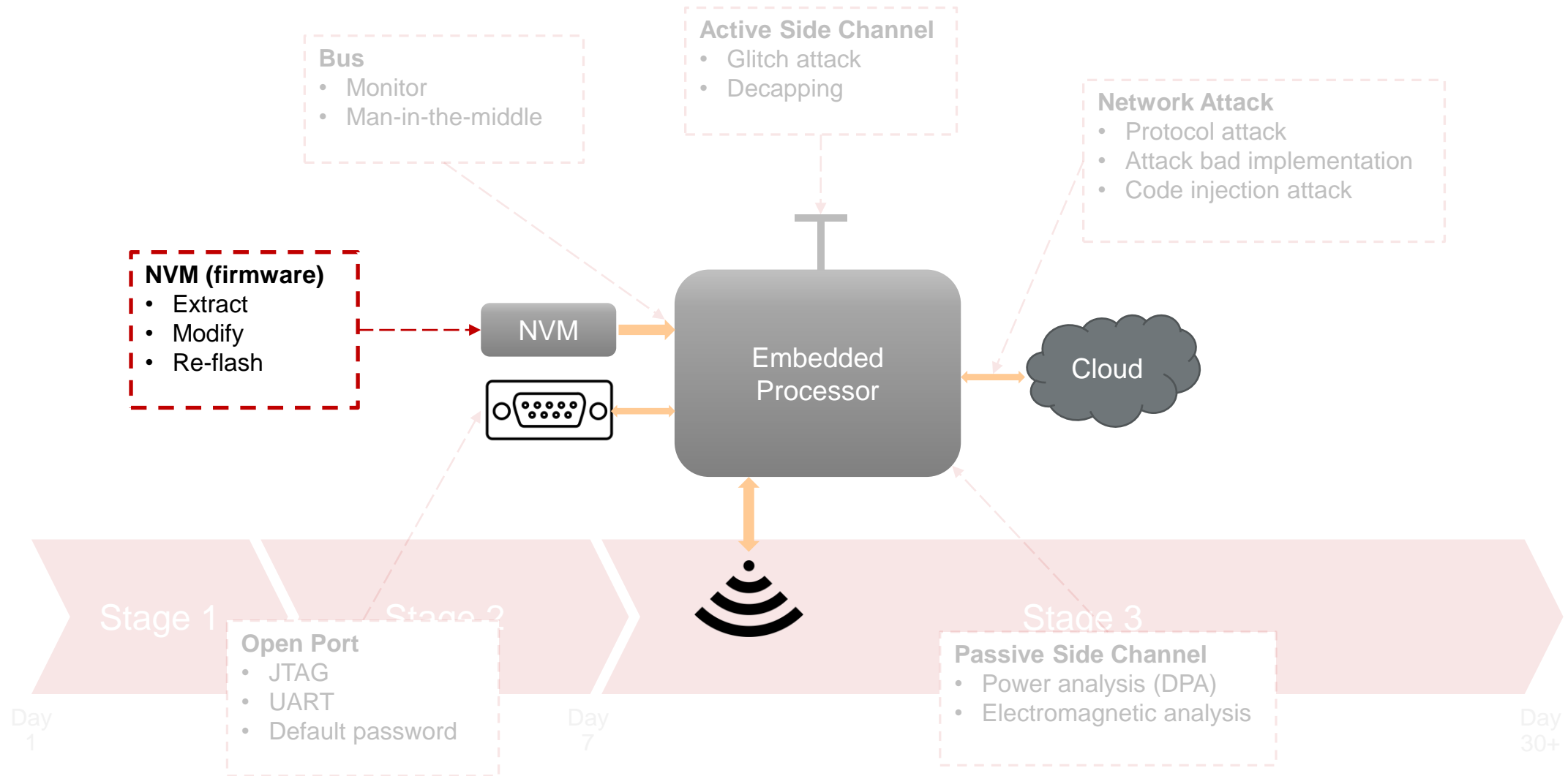
- Prevents brute force attack. For example, if OPTIGA™ Trust M performs 1 decryption (secret key use), 1 signature generation (private key use), 1 key derivation **within t<sub>max</sub>**, then SEC = (1+1+1) = 3. After t<sub>max</sub> elapses, SEC = (3-1) = 2
- **Throttling down profile**  
OPTIGA™ Trust M throttles its performance as a guard against planned attacks like side channel analysis involving power. OPTIGA™ Trust M starts slowing down the operations by inducing a specific amount of delay.

...



Link for more information: **OPTIGA™ Trust M: Security monitor – KBA235349**

# Firmware Access MUST Be Protected





# Use PSoC™ Secure Boot to Only Run Genuine Software

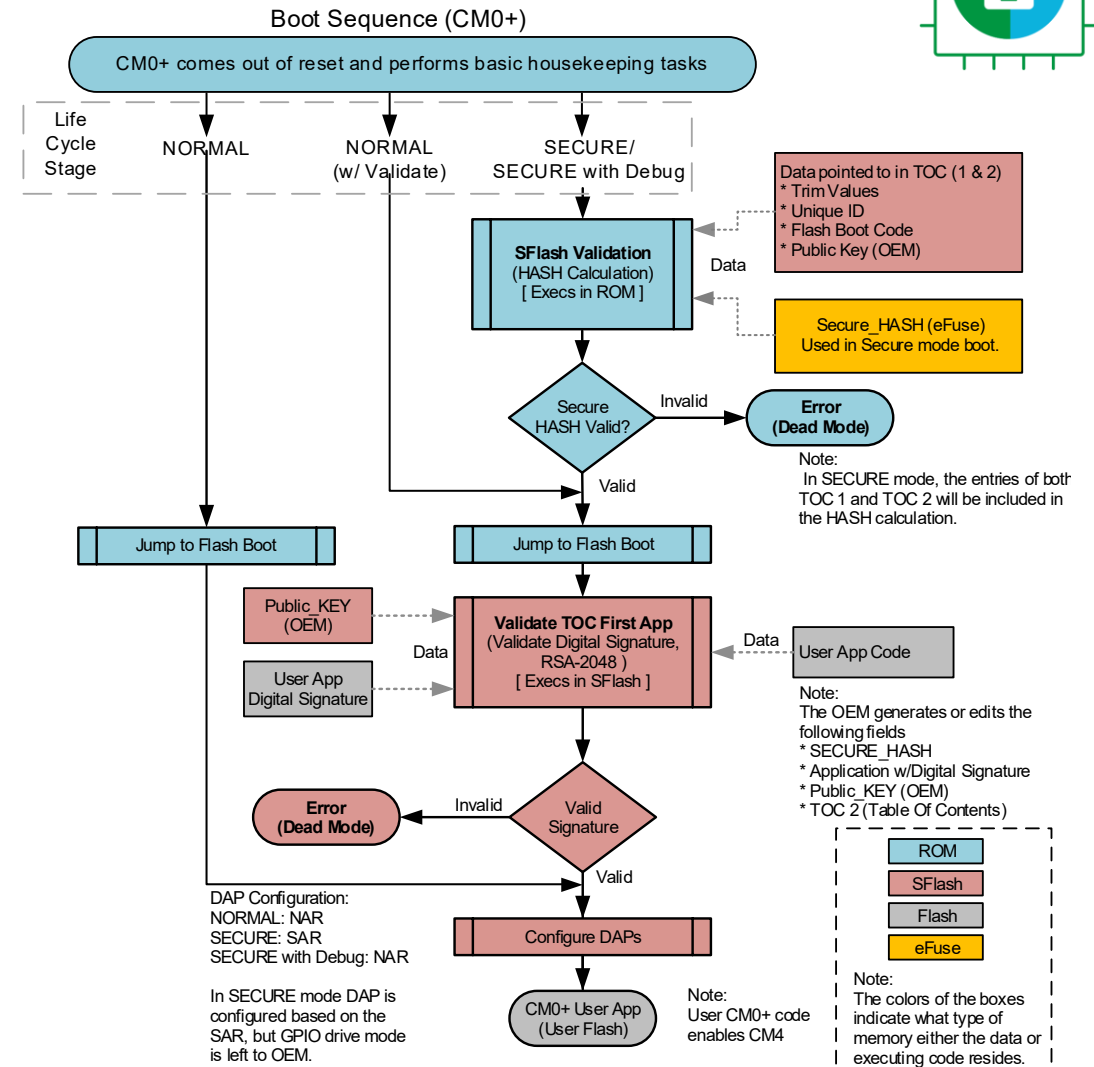


**Secure Boot** must be used to ensure that only manufacturer firmware can run on the MCU.

To verify the user application, a digital signature is created and appended to the end of the code during build time.

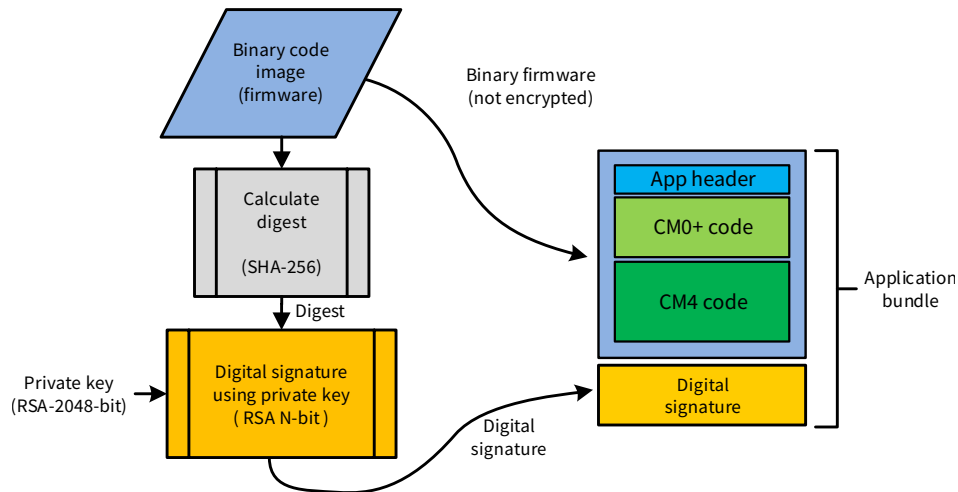
During the Boot Sequence, first the TOC1&2 data (Trim Values, Unique ID, Flash Boot Code, Public Key (OEM)) are being validated using digest stored in eFuse.

In second step User App Code is being validated using SFlash data.

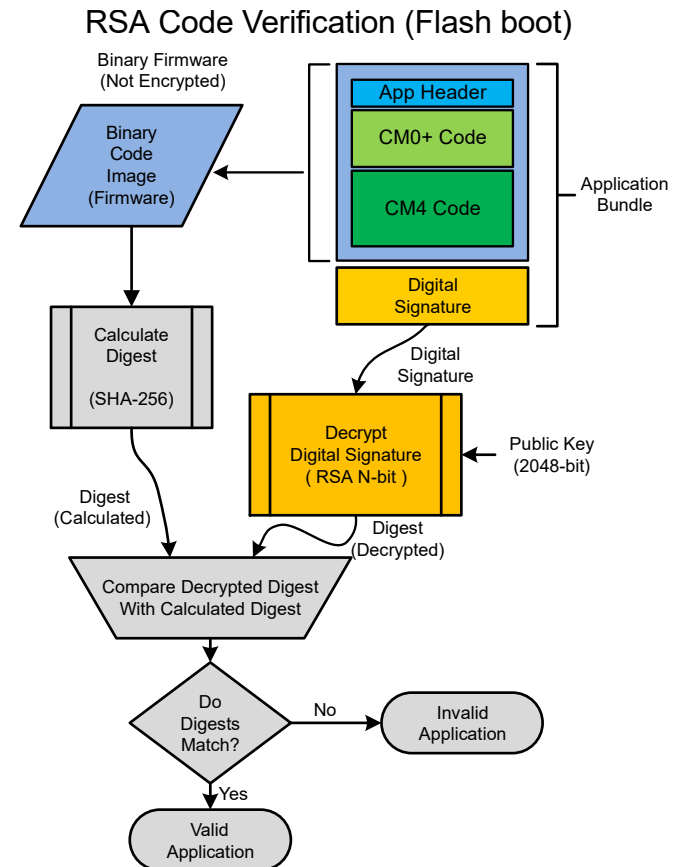


# Code Signing & Code verification

To verify the user application, a digital signature is created and appended to the end of the code during build time.



The device bootup code “Flash boot” uses RSA to verify first the user code.



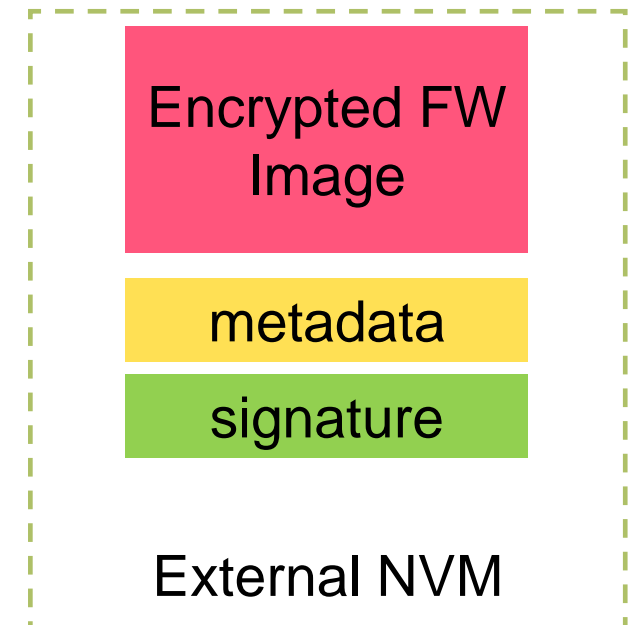
# Use Memory Encryption to Protect Firmware Stored Externally

When using external NVM to store the firmware, **flash encryption** is recommended. This way physical readout of the external SPI flash memory is **not** sufficient to recover most flash contents, thus protecting firmware against **unauthorized readout, reversing** or **modification**.

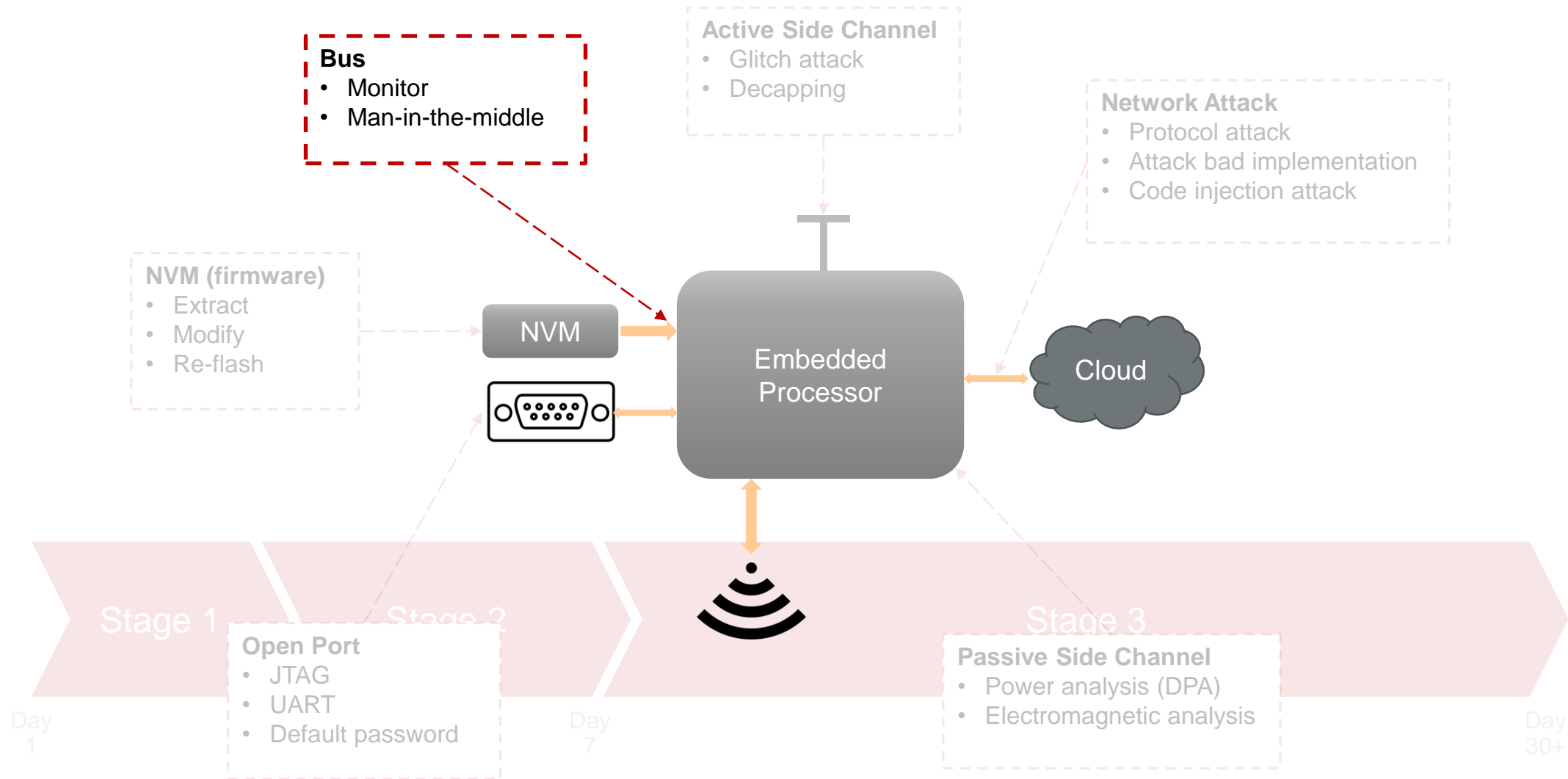
The content consist of:

- Encrypted FW image
- Metadata:
  - Version → Roll back prevention
  - Image Hash → Image integrity
  - Derivation Data → Binding metadata with Image & Confidentiality
  - Known Good Platform Status → Update platform integrity reference on Host MCU/Trust M as data object
- Signature – encrypted digest using public key of device

Metadata is „personalised“ preventing physical attack by changing the external NVM between devices



# Bus Monitoring Recommendations



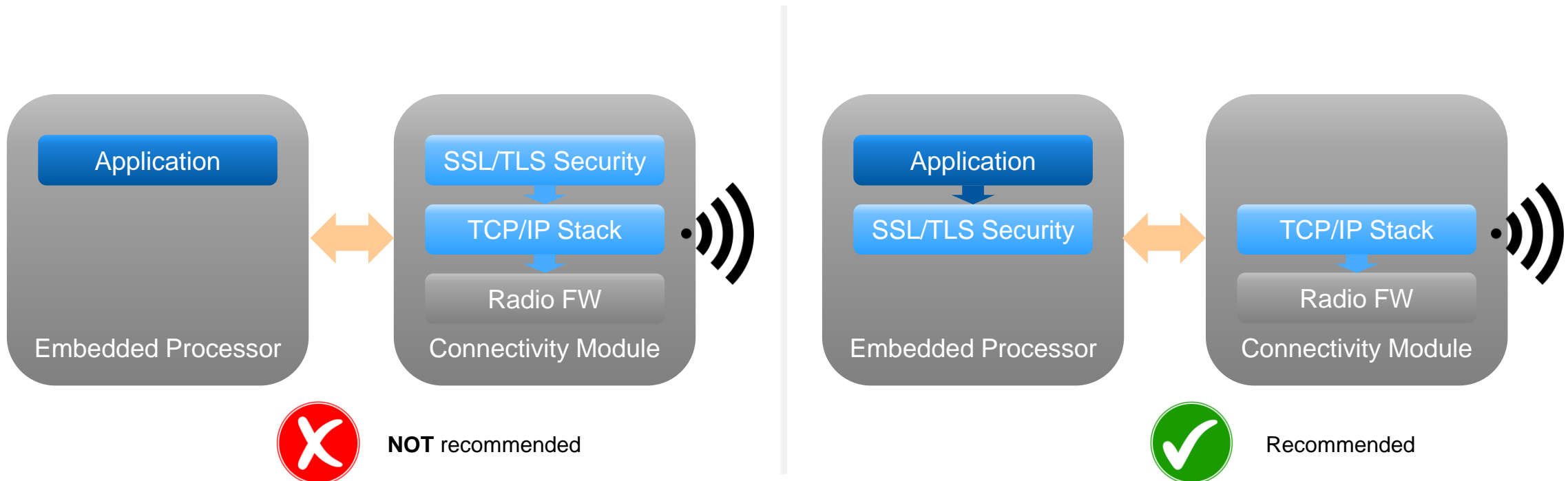


# Bus Monitoring Recommendations

**Use peripheral that supports bus encryption whenever it is possible.**

If encryption at bus level is not possible, at least **use encryption at application level**.

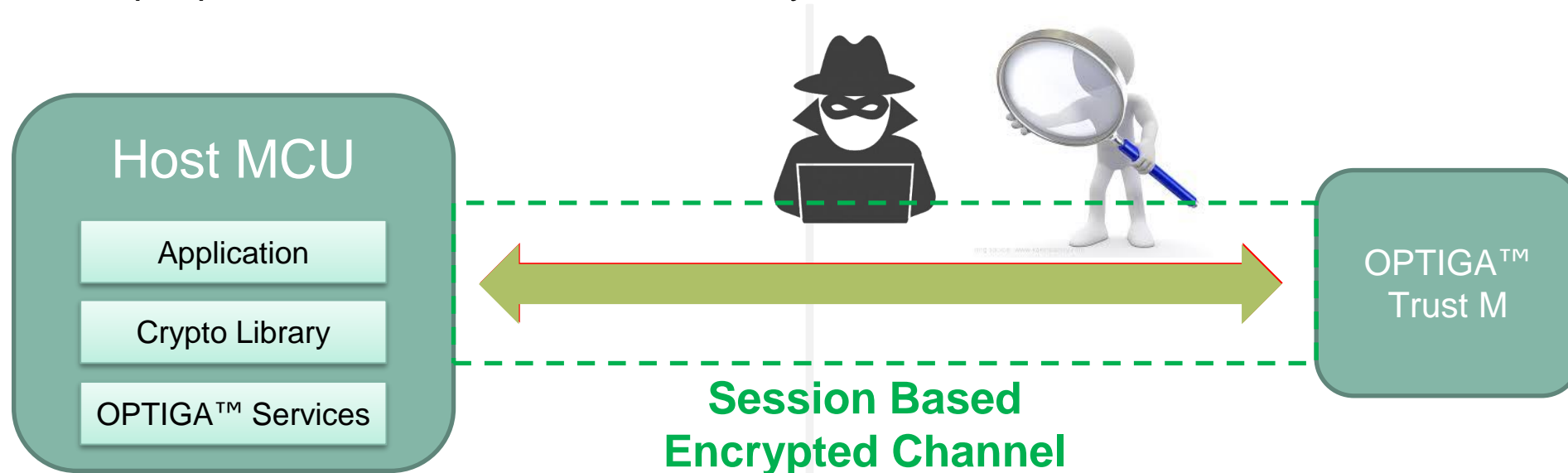
Connectivity modules that include TLS stack in module firmware are at risk if data bus is not encrypted; as protocol and application data is visible in clear on the bus and replay attack can be performed.



# OPTIGA™ Shielded Connection – Prevent spying of the I2C bus

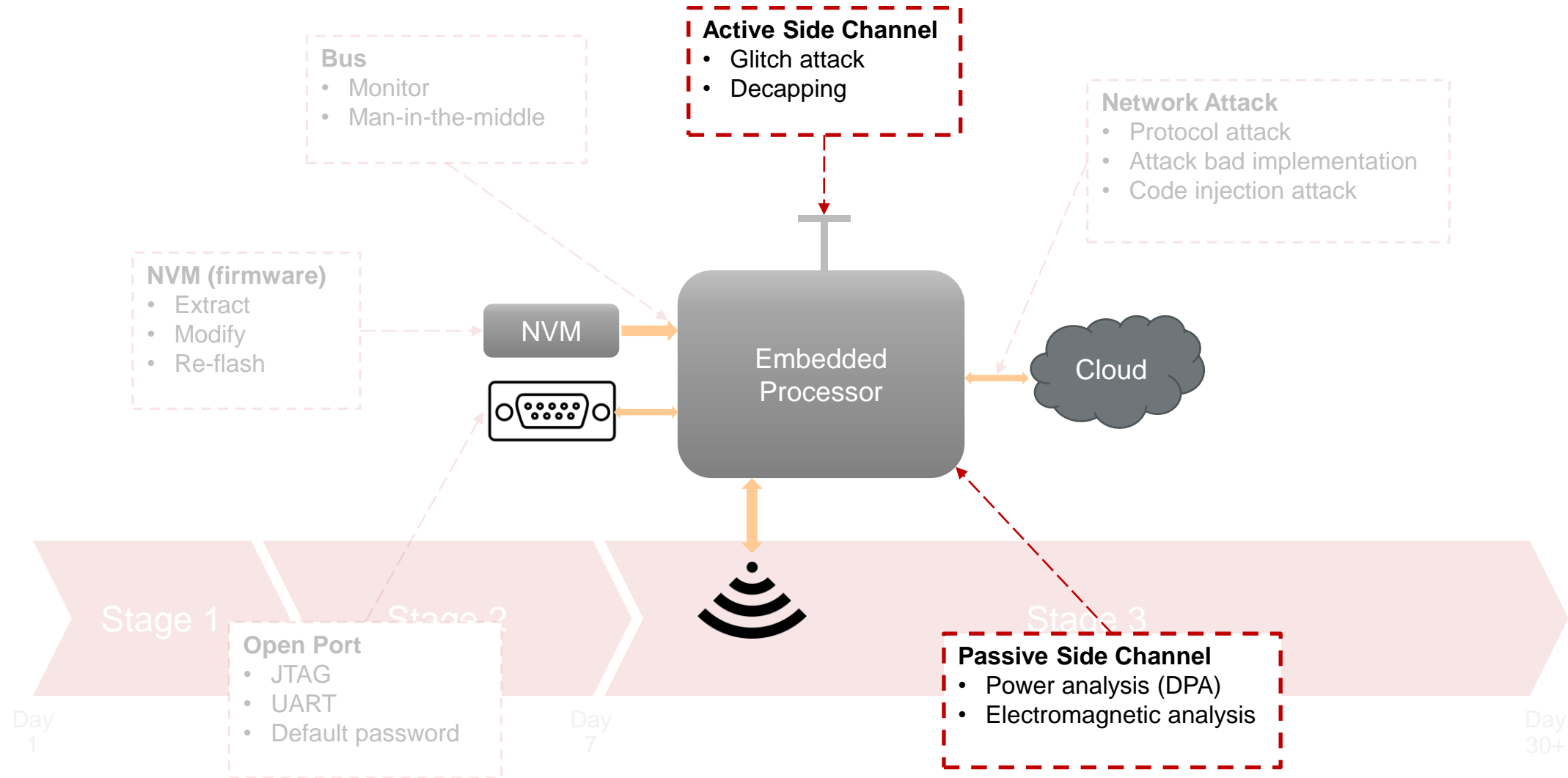
The OPTIGA™ Shielded Connection enables a protected (Integrity and Confidentiality) communication between the OPTIGA™ and a corresponding Host platform. The pre-shared secret is established during first boot/initialization sequence.

The session key is derived every time a shielded communication is established between the host and OPTIGA™ Trust M and therefore, unique on each startup or each session. This makes the symmetric key used unique per session and enhances security, similar to TLS.



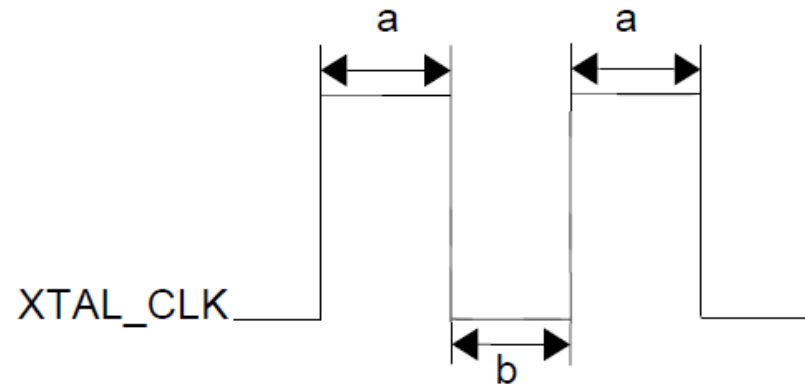
Link for more information: **OPTIGA™ Trust M: Shielded connection – KBA235350**

# Side Channel Attacks



## Clock Glitch Detection (Espressif C3/S3)

The **Clock Glitch Detection** module monitors input clock signals from **XTAL\_CLK**. If it detects a glitch, namely a clock pulse (a or b in the figure below) with a width shorter than 3 ns, input clock signals from XTAL\_CLK are blocked.



Once detecting a glitch on XTAL\_CLK that affects the circuit's normal operation, the Clock Glitch Detection module triggers a **system reset** (including RTC) if RTC\_CNTL\_GLITCH\_RST\_EN bit is enabled. By default, this bit is set to enable a reset.



# Side Channel Attacks

**Most general purpose MCU on the market are still vulnerable to side channel attacks such as **decapping** or **Differential Power Analysis (DPA)**.**

Decapping a chipset is **not very expensive** and definitely in reach for hackers.

**Device cloning is a real risk.**

- Provided enough time, decapping technique has a close to 100% success rate to **extract secret keys** and **device firmware**.
- Copying the device BoM is no big challenge.



# Use SE against Side Channel Attacks

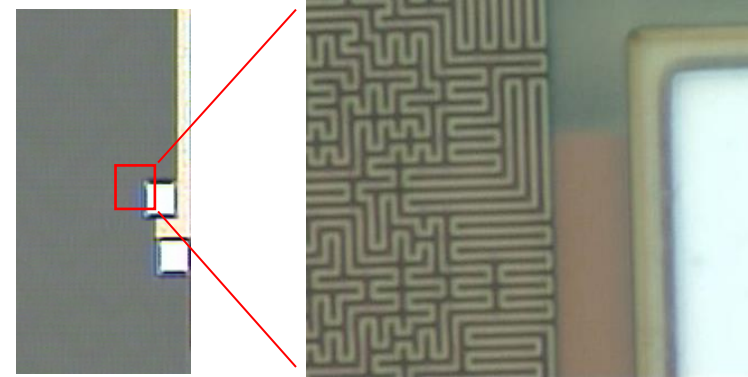
## Advanced Multi-Level Hardware Security

- Active shield over entire chip
- All memories internally encrypted
- Information independent execution
- Internal state consistency checking
- Power supply tamper protection
- Temperature lockouts
- Internal clock generation
- Secure test methods
- No die features can be identified
- No package or die identification

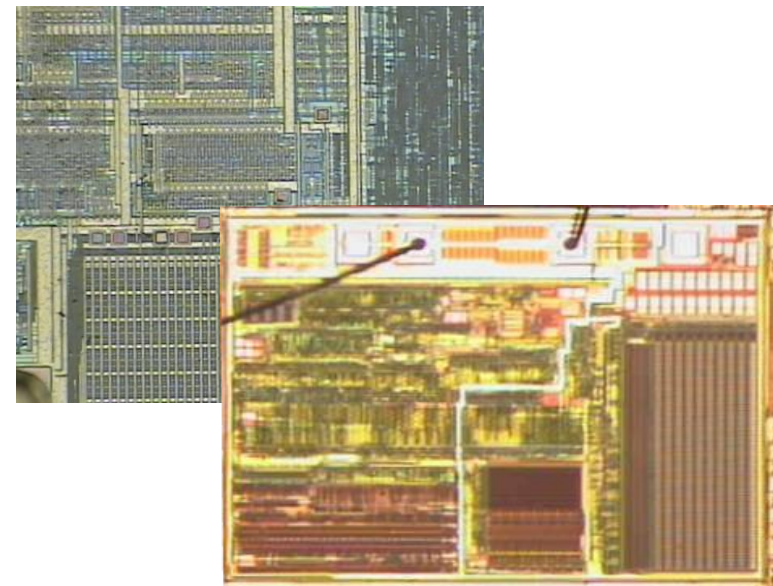
## Designed to defend against a multitude of attacks

- Legacy architecture from Smart Card history

Active Shield

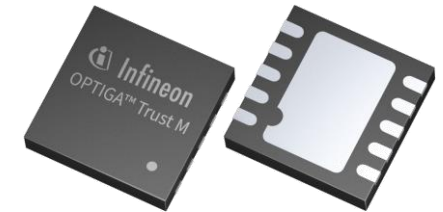


Standard MCU, logic & memory

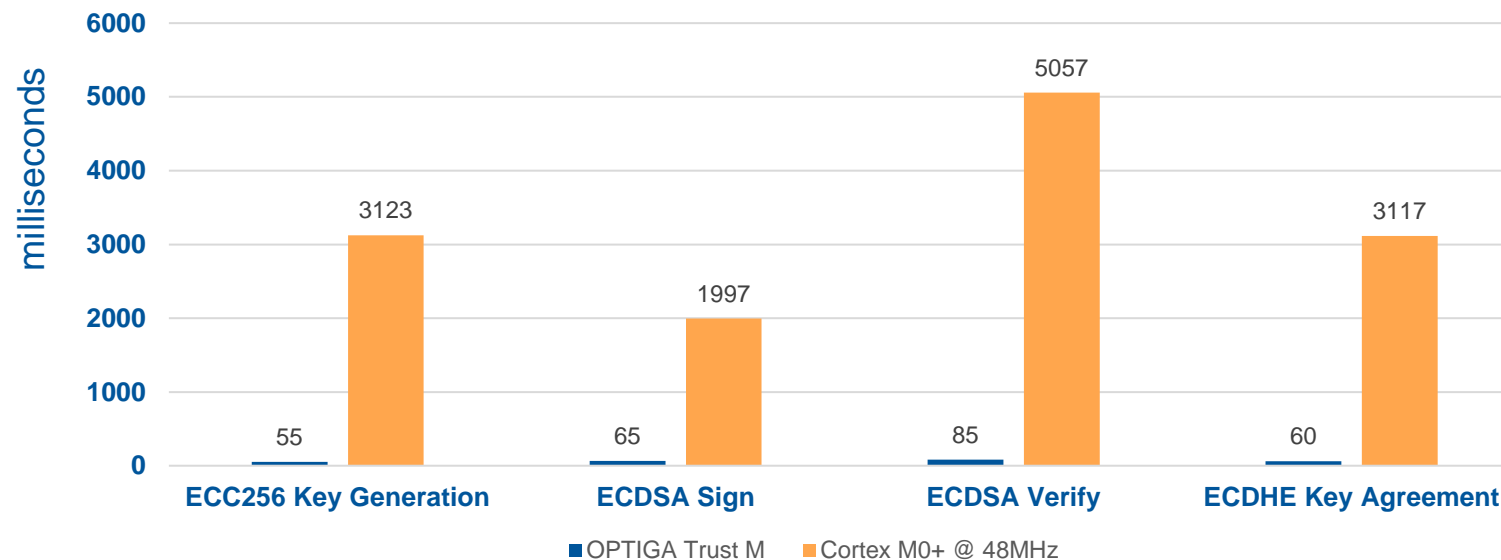


# OPTIGA™ Trust M – Crypto performance

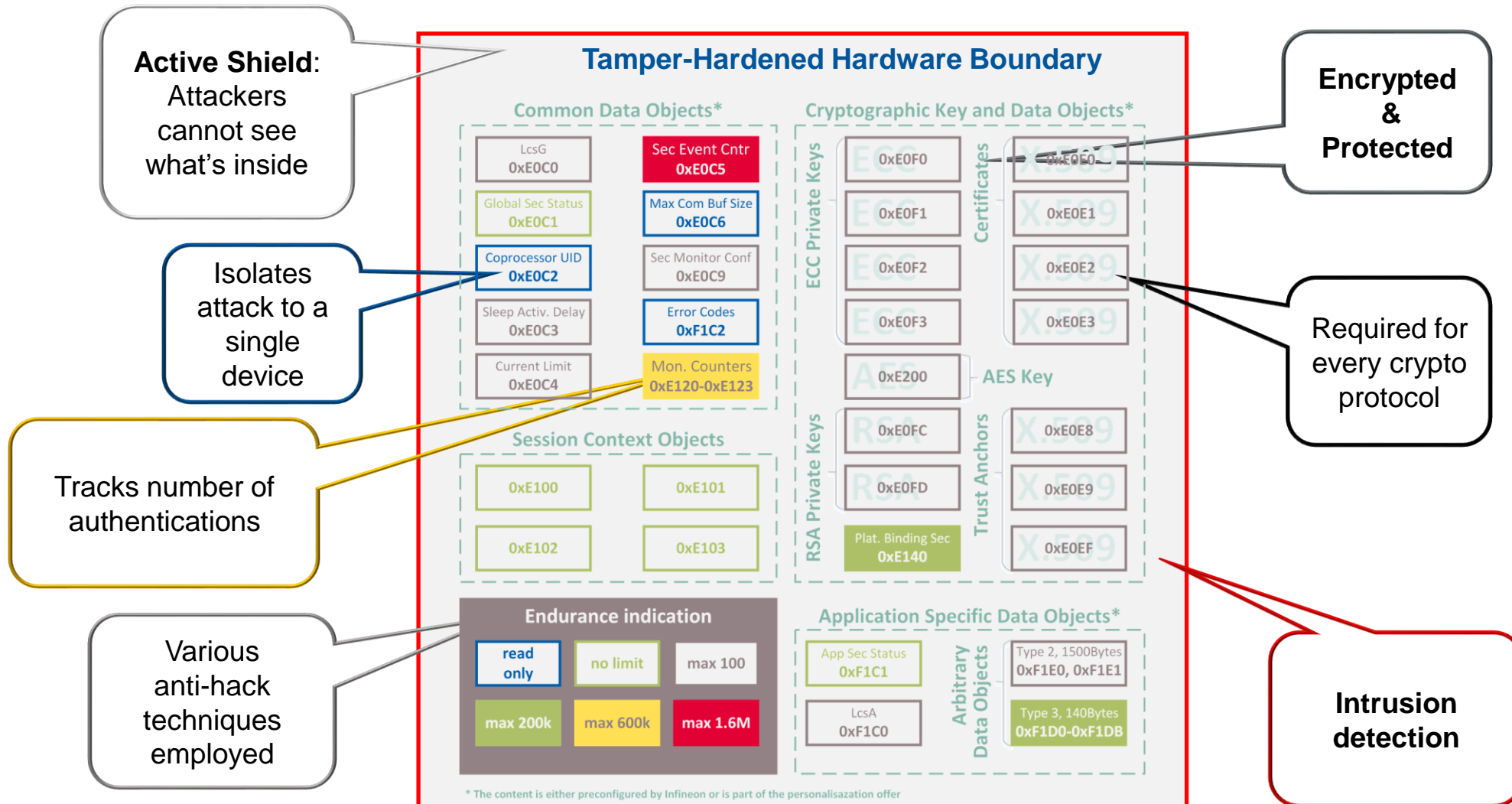
- Supports Elliptic Curve Cryptography authentication (**ECDSA**) and key exchange (**ECDH**)
- Encryption (**AES**) is still handled by the host MCU (data bus is encrypted)
- **Protects** the device's identity key
- **Accelerates** verification and key agreement



## OPTIGA™ Trust M vs. Cortex-M0+ (SW) Benchmark

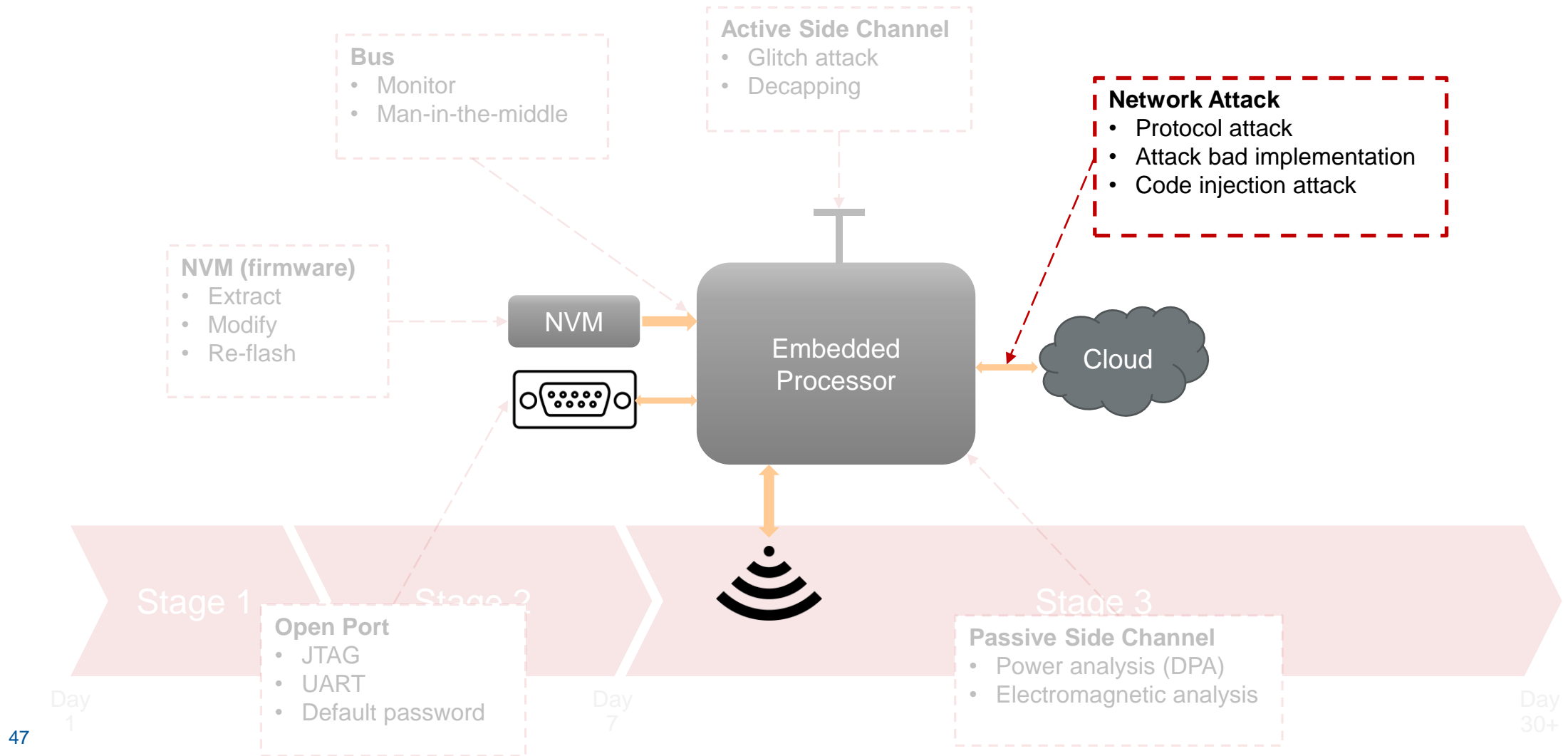


# Use SE against Side Channel Attacks – OPTIGA™ Trust M





# Network and Software Attacks



# Use Canary to Detect Code Injection Attacks (GCC)

To counter such exploit, compilers including the **gcc** started to add 'hardening' options to detect these exploits. In that approach, the compiler is placing a '**canary**' guard into each instrumented function **stack frame**.

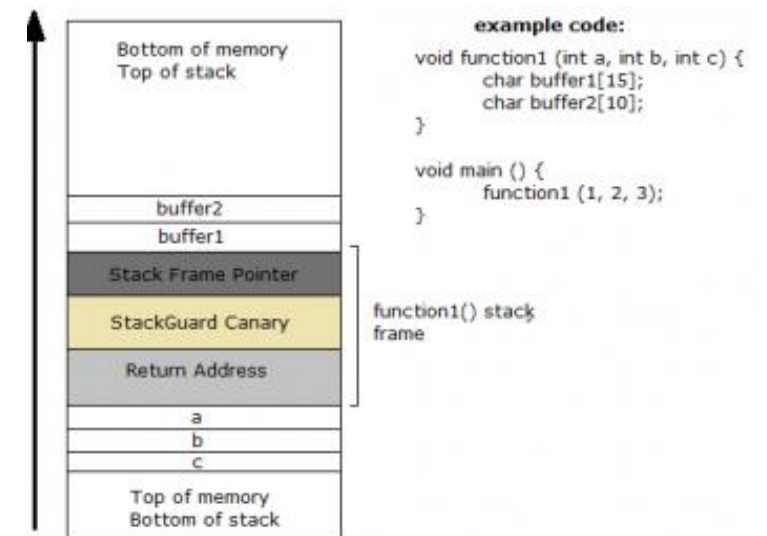
**-fstack-protector**: Emit extra code to check for buffer overflow, such as stack smashing attack. This is done by adding a guard variable to functions with vulnerable objects. This includes functions that call `alloca`, and functions with buffers larger than 8 bytes. The guards are initialized when a function is entered and then checked when the function exits. If a guard check fails, an error message is printed and the program exits.

**-fstack-protector-all**: Like `-fstack-protector` except that all functions are protected.

```
1 unsigned long __stack_chk_guard = 0xDEADBEEF;
2
3 void __stack_chk_fail(void) { /* will be called if guard/canary gets corrupted */
4     /* Handle error, print error message, stop the target, ... */
5     DisableInterrupts();
6     __asm volatile("bkpt #0"); /* break target */
7 }
```

The compiler then generates the code verification logic:

- At function entry, it stores the **\_\_stack\_chk\_guard** value into the stack frame.
- At function exit, the guard value on the stack is compared against the value in **\_\_stack\_chk\_guard**. If canary is overwritten, handler **\_\_stack\_chk\_fail** is called.



# Best practices to Prevent Code Injection Attacks - Validate Input

One of the most important steps to prevent code injection attacks is to validate the input that your application receives from users or other sources. Validation means checking that the input conforms to the **expected format, type, length, and range, and rejecting any input that does not**. You can use built-in or custom validation functions, regular expressions, or whitelists to filter out any unwanted or suspicious input. Validation should be performed on both the client-side and the server-side, as client-side validation can be bypassed by malicious users.

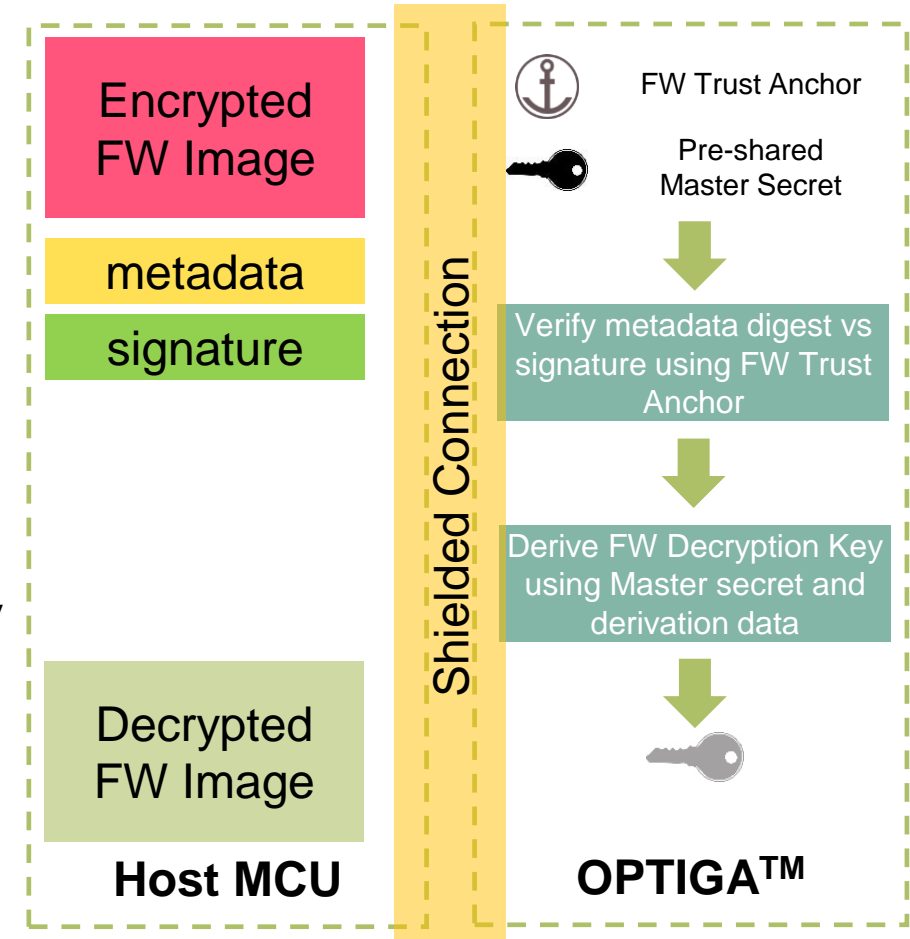
```
#define MAX_INPUT_BUFFER 128
uint8_t input_buf[ MAX_INPUT_BUFFER + 1];

while(1){
    input_buf[current_byte] = input_data;
    ...
    if(current_byte < MAX_INPUT_BUFFER) current_byte++;
}
```

# Secure (OTA) Update using OPTIGA™ Trust M

## SoC / MCU FW

- The SoC / MCU FW gets distributed using a multicasting model.
- The FW Update Data Set consists of update metadata and the update image.
- The Integrity of the update metadata roots back to the FW Update Root CA (Trust Anchor). The regarded public key certificate hosted by the OPTIGA™ Trust M.
- The update metadata consists among others of
  - Version → Roll back prevention
  - Image Hash → Image integrity
  - Derivation Data → Binding metadata with Image & Confidentiality
  - Known Good Platform Status → Update platform integrity reference on Host MCU/Trust M as data object
- The derived image decryption key roots back to a shared secret known by all OPTIGA™ Trust M of a given Platform Vendor domain.





**TECHNOLOGY.**

**PASSION.**

**EBV.**

