

# USB ARMORY RELOADED

Andrea Barisani – Head of Hardware Security



\$ whoami

Andrea "lcars" Barisani

I am a



Founder of **INVERSE**  **PATH** now part of 

Breaking things since I got my first



Securing     and much more since 2005.

Maker of the USB armory



Speaker and trainer at BlackHat, CanSecWest, DEFCON, Hack In The Box, PacSec conferences among many others.

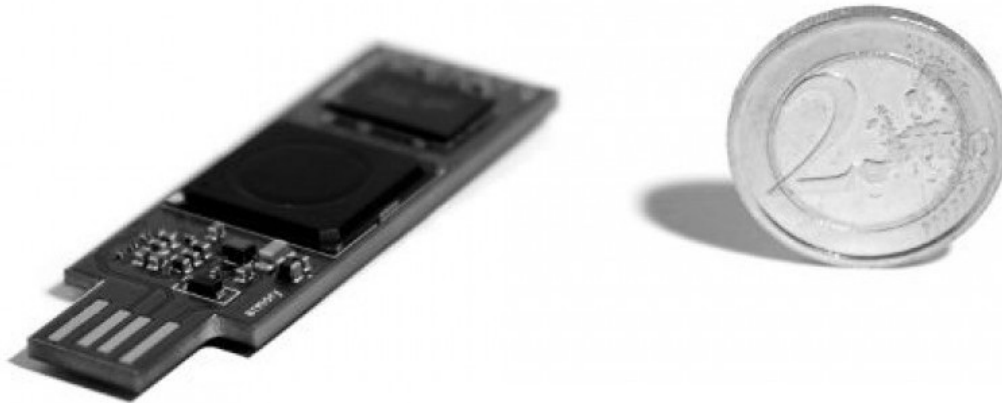
<https://andrea.bio> | @andreabarisani

**October 15<sup>th</sup> - Hack In The Box 2014 – Kuala Lumpur**



# **Inverse Path Announce Armory SoC Project**

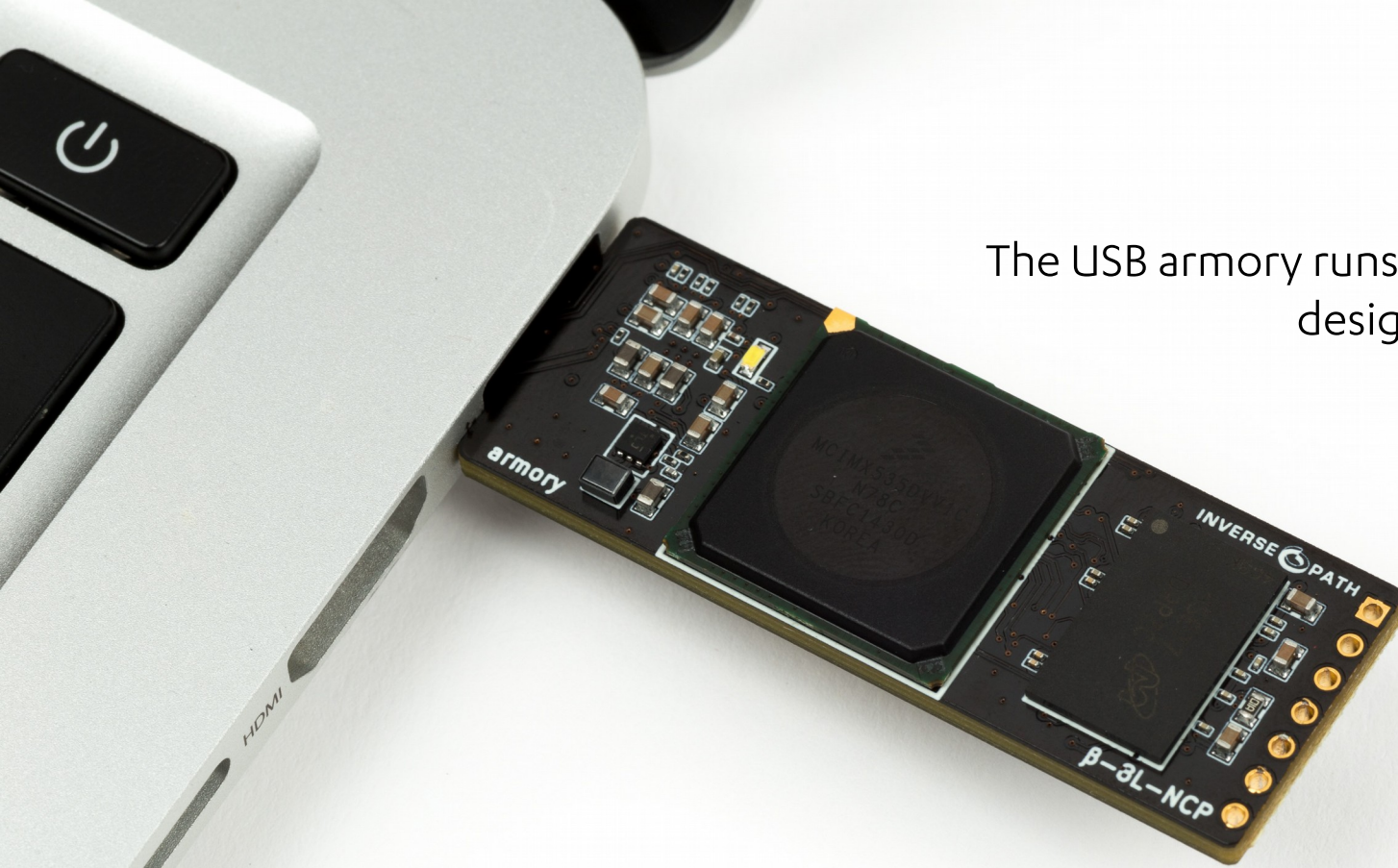
posted on **OCTOBER 3, 2014** by **L33TDAWG**



The USB armory runs Linux and is a personal server designed for security applications:

- Password manager
- Encrypted storage
- Authentication token
- Cryptocurrency wallet
- Secure messaging
- Hardware Security Module

Customers range from individual security researchers, security companies to large enterprises and government entities.



# USB armory Mk I



One of the smallest SBC in the world, met with outstanding demand from security researchers, businesses, OEMs, integrators and security companies.

## The good stuff...

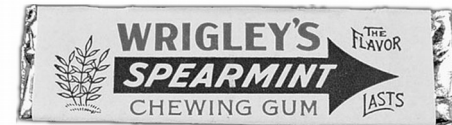
It wasn't easy to fill the support gap left by NXP, but we did it until we hit the actual hardware and this resulted in several OSS contributions.

Form factor, priority on security and transparency, the incredible projects and use cases we never dreamed of.

Great research platform for all things (e.g. TrustZone).

## MACGYVER ARMORY

### 20th Century



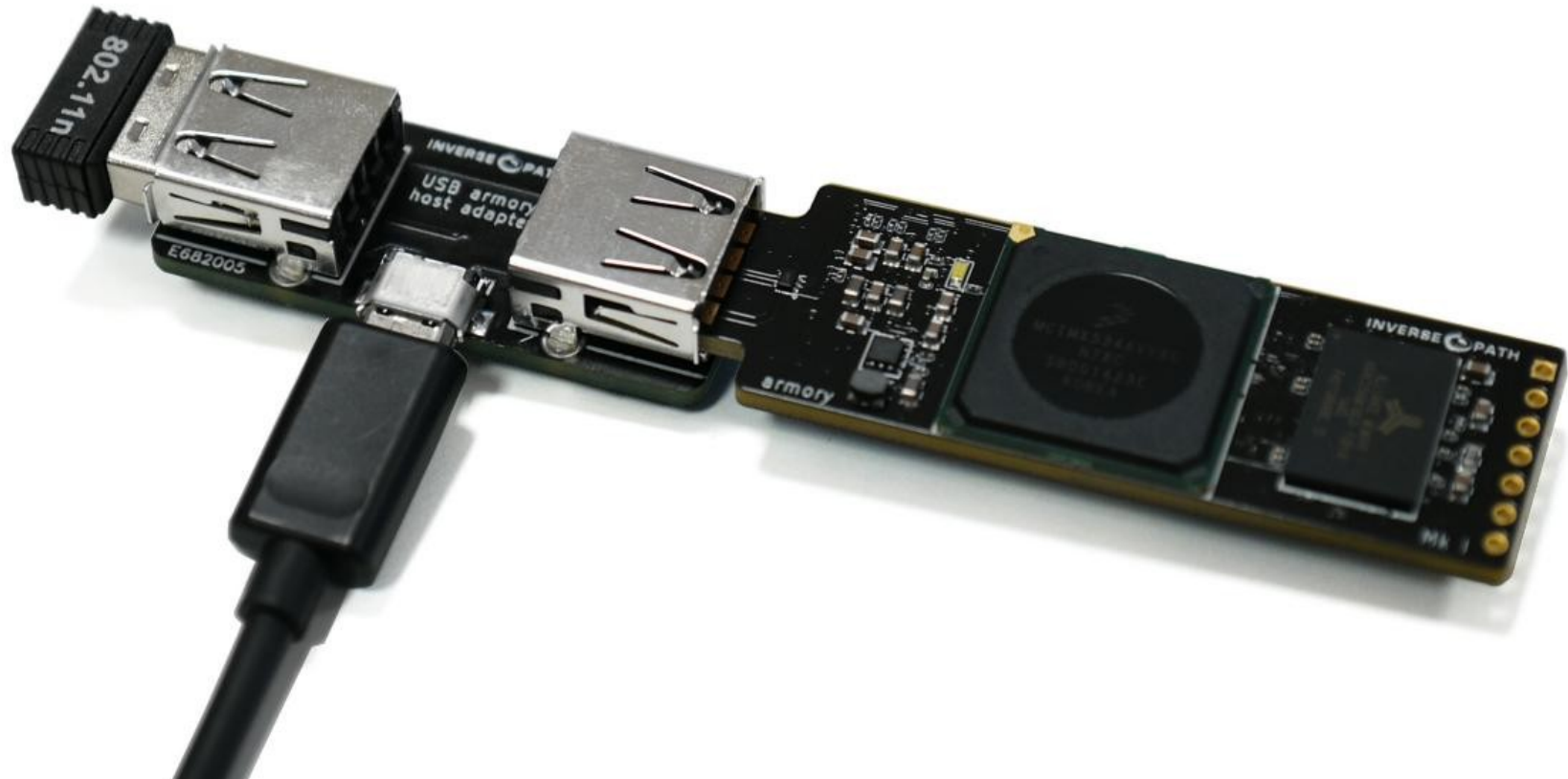
DNA storage, conductor, sealing material, adhesive, stress relieve, nomnom  
76mm x 19mm

### 21st Century



Open source hardware and software, ARM Cortex-A8 800MHz, 512MB RAM, microSD, USB 2.0 OTG, Ethernet/storage/UART/HID/etc device emulation, 65mm x 19mm

# USB armory Mk I



## Errata...

The microSD hinge is “challenging”.

The PCB plug, in retrospect, was a bad call.

Designing enclosure as an afterthought is a nightmare.

We learned the hard way that NXP long term support does not entail security.

Lack of built-in storage restricts provisioning scalability.

Not ideal for general consumer applications.



# HABv4 bypass



In 2017 Quarkslab discovered critical security vulnerabilities that affect HABv4 on the entire NXP i.MX series.

The issue was reported for the i.MX6, Inverse Path immediately investigated applicability to the i.MX53.

An X.509 parsing error (ERR010873 | CVE-2017-7932) and an SDP protection bypass (ERR01872 | CVE-2017-7936) allow arbitrary code execution on SoC in Closed configuration.

The findings prevent the secure operation of unattended setups while attended setups remain protected in case of device loss (but not tampering).

NXP did not release any P/N updates for the i.MX53.

[https://github.com/inversepath/usbarmory/blob/master/software/secure\\_boot/Security\\_Advisory-Ref\\_QBVR2017-0001.txt](https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_QBVR2017-0001.txt)



# HABv4 bypass



## Timeline

=====

2017-05-18: Quarkslab presents findings at the 2017 Qualcomm Mobile Security Summit [9], materials are not disclosed to the public at this time.  
2017-05-30: Quarkslab communicates embargo period until 2017-07-18.  
2017-05-30: Inverse Path proposes preliminary advisory release on 2017-06-05.  
2017-06-05: Inverse Path releases preliminary advisory.  
2017-06-06: added assigned CVE numbers.  
2017-07-19: Quarkslab public release of findings [4].  
2017-07-19: Inverse Path release of full advisory and i.MX53 PoC [6].  
2017-07-27: added link to i.MX Community post that lists affected P/Ns.

F-Secure prioritized announcing the existence of the issue before the full advisory release, additionally developed and released a full PoC.

The `usbarmory_csftool` is the only Open Source implementation for HABv4 signing as well as the first and only exploitation tool ;-)

“Break your own product, and break it hard”

<https://labsblog.f-secure.com/2017/07/19/break-your-own-product-and-break-it-hard/>

# HABv4 bypass



```
$ usbarmory_csftool -h
Usage: usbarmory_csftool [OPTIONS]
-A | --csf_key <private key path> CSF private key in PEM format
-a | --csf_cert <public key path> CSF public key in PEM format
-B | --img_key <private key path> IMG private key in PEM format
-b | --img_cert <public key path> IMG public key in PEM format
-I | --table <SRK table path> Input SRK table (see usbarmory_srkttool -0)
-x | --index <SRK key index> Index for SRK key (1-4)
-i | --image <filename> Image file w/ IVT header (e.g. u-boot.imx)
-o | --output <filename> Write CSF to file
-s | --serial Serial download mode
-S | --dcd <address> Serial download DCD OCRAM address
    (depends on mfg tool, default: 0x00910000)

-d | --debug Show additional debugging information
-T | --hab_poc Apply HAB bypass PoC (CVE-2017-7932)

-h | --help Show this help
```

Publishing PoC code encourages further investigation and testing of issues among vendors or other affected parties; it promotes security research; and it empowers other skilled parties to further verify the scope and impact of vulnerabilities.

The most important and compelling reason to take this approach, however, is this: In scenarios where detailed technical information has already been made public, the lack of a working PoC does not, and should not, constitute any form of “protection.”

# The future



The **USB armory Mk II** aims to continue our support for this class of product and to improve the Mk I.

## Primary design goals

The microSD hinge replacement with a push/pull slot.

Real USB plugs, plug + socket for integrated host adapter.

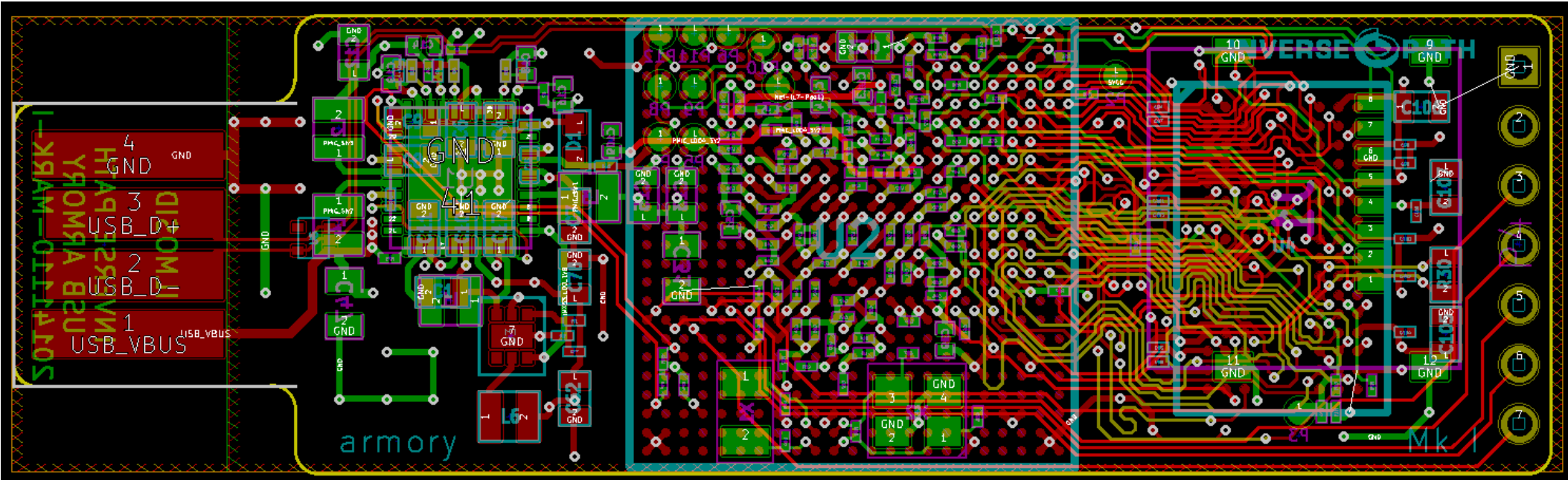
Enclosure design right from the beginning.

Full internal and third party security audit for HABv4 and chain of trust.

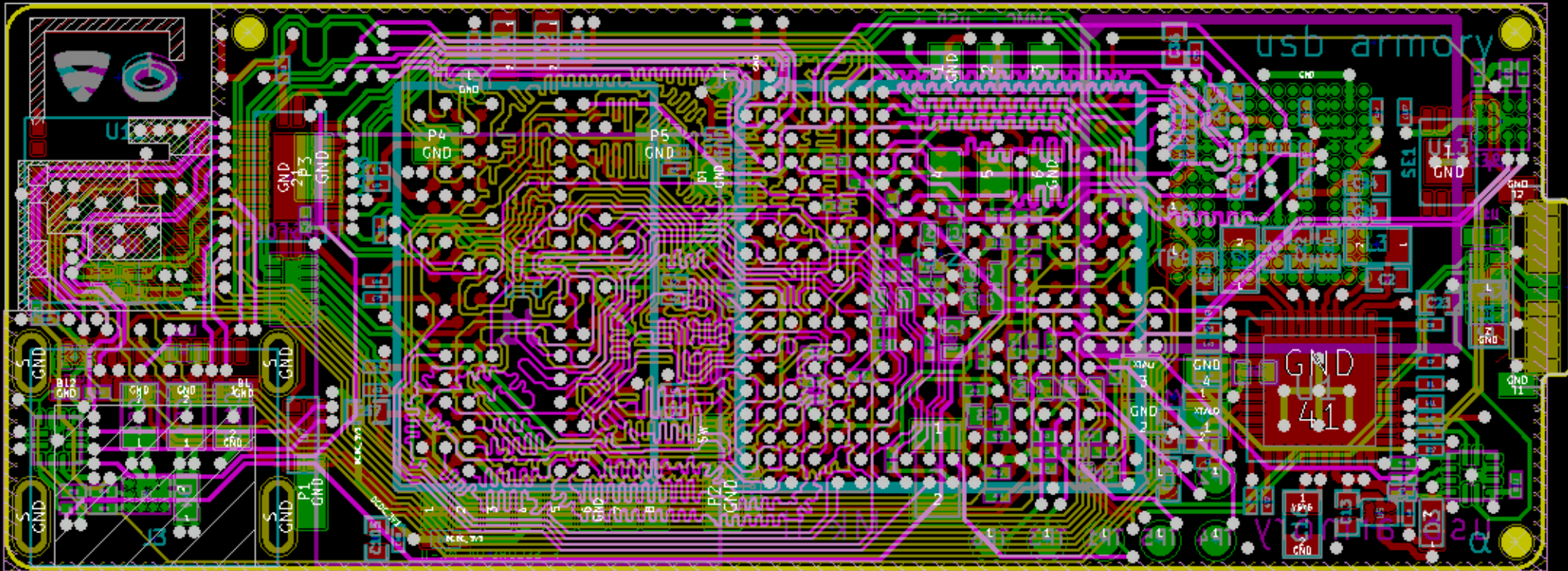
Addition of built-in eMMC storage and external crypto authenticator.

Bluetooth communication.

# USB armory Mk I

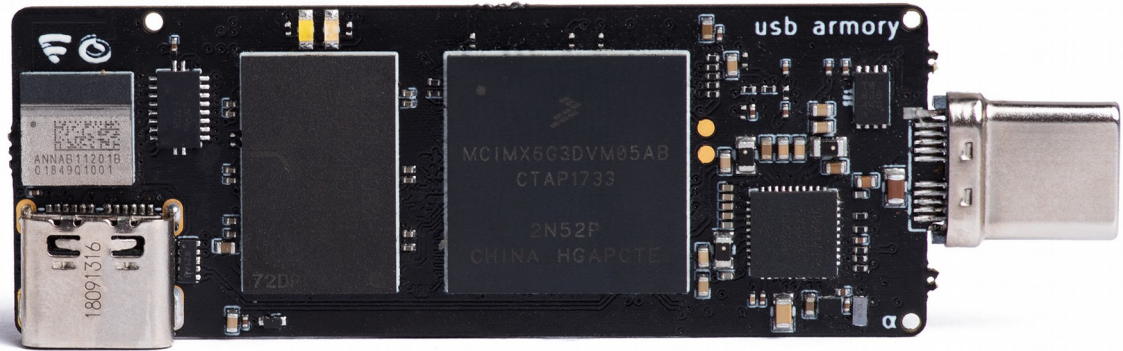


# USB armory Mk II



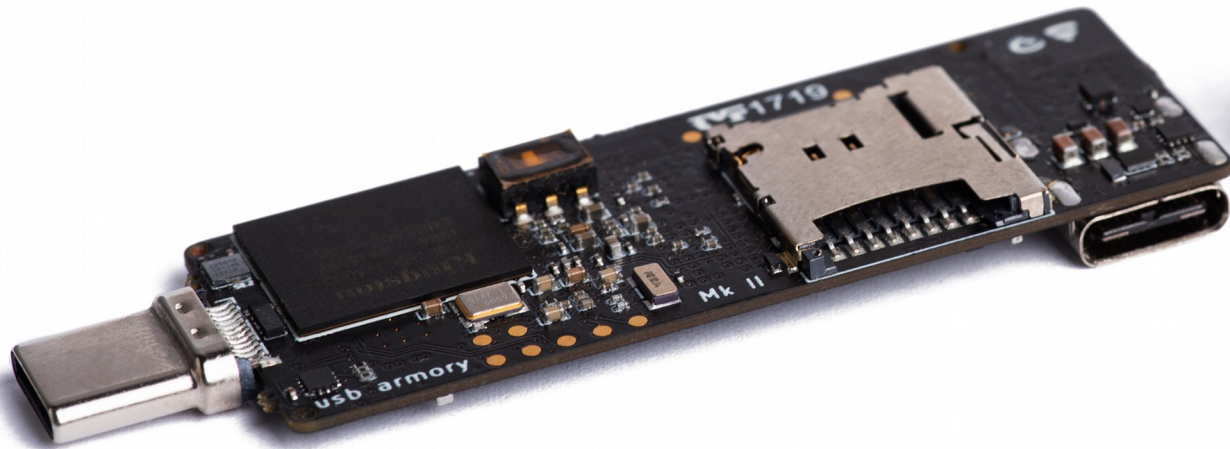
open source  
hardware

# USB armory Mk II



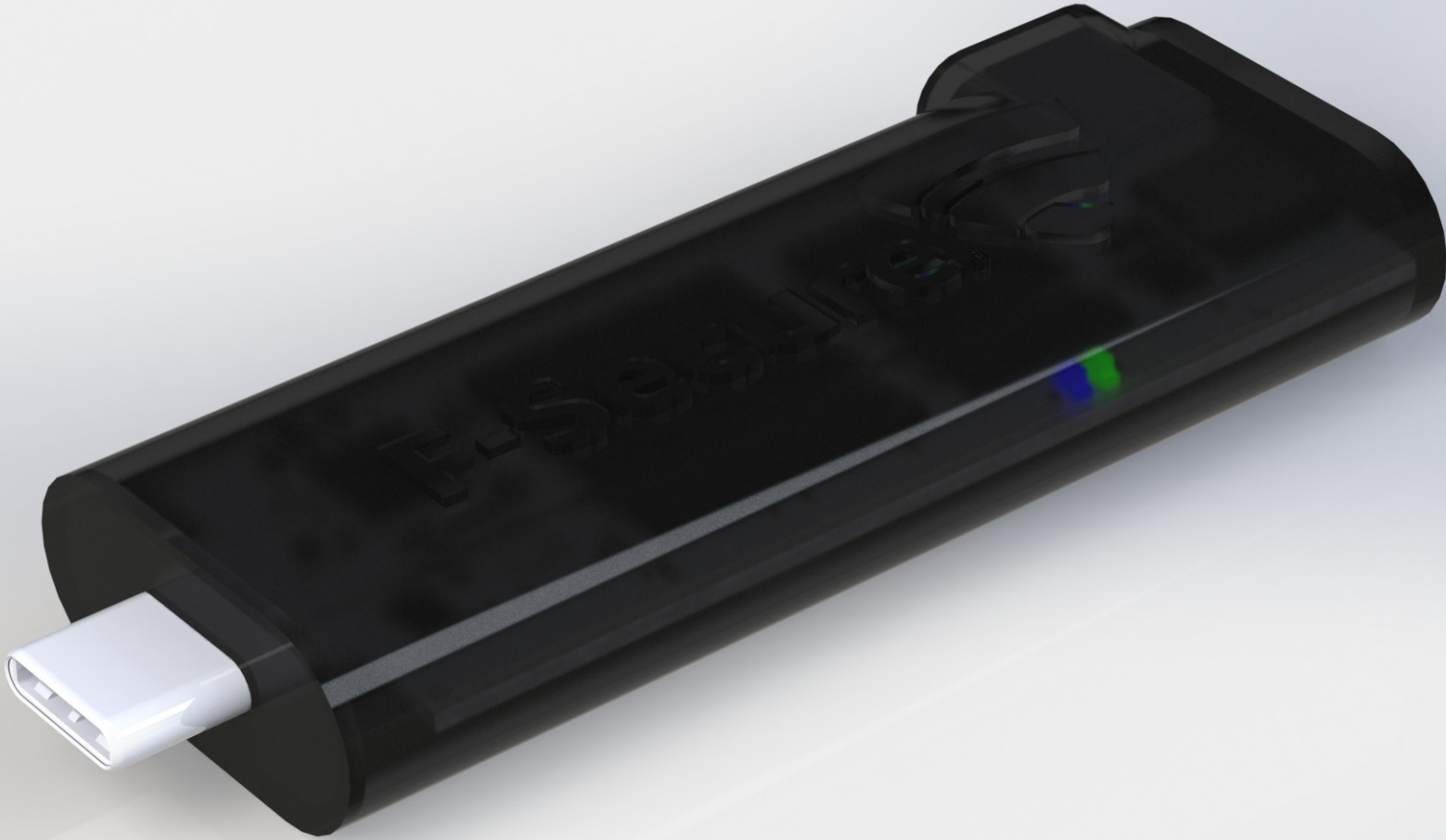


# USB armory Mk II





## Enclosure design



## Enclosure design



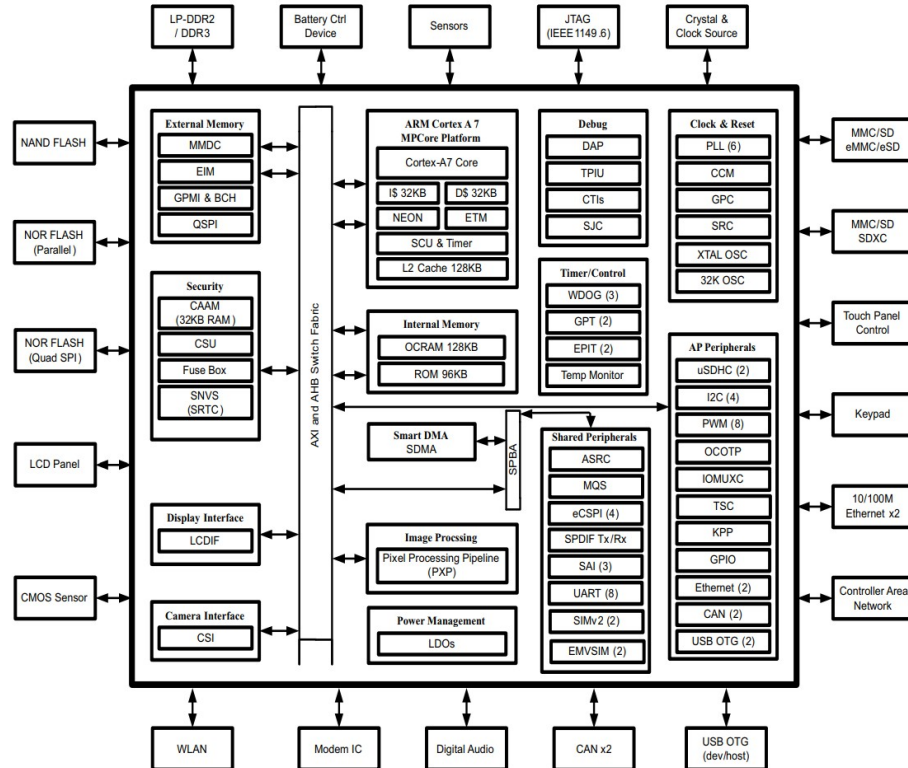
## Enclosure design



# NXP – i.MX6UL /i.MX6ULZ



ARM® Cortex™-A7 528/900 MHz



Hardware security features

High Assurance Boot (HAB 4.2.6)

Cryptographic accelerator (CAAM/DCP)

True Random Number Generator (TRNG)

Bus Encryption Engine (BEE - OTF AES) (i.MX6UL only)

Secure Non-Volatile Storage (SNVS)

ARM® TrustZone®



# NXP - Secure Non-Volatile Storage (SNVS)



The SNVS feature relies on the OTPMK, a **unique per-device hardware key**, which cannot be read directly as it can only be used via the SoC internal Cryptographic Accelerator and Assurance Module (CAAM), when secure booted.

The SNVS feature can be summarized as follows:

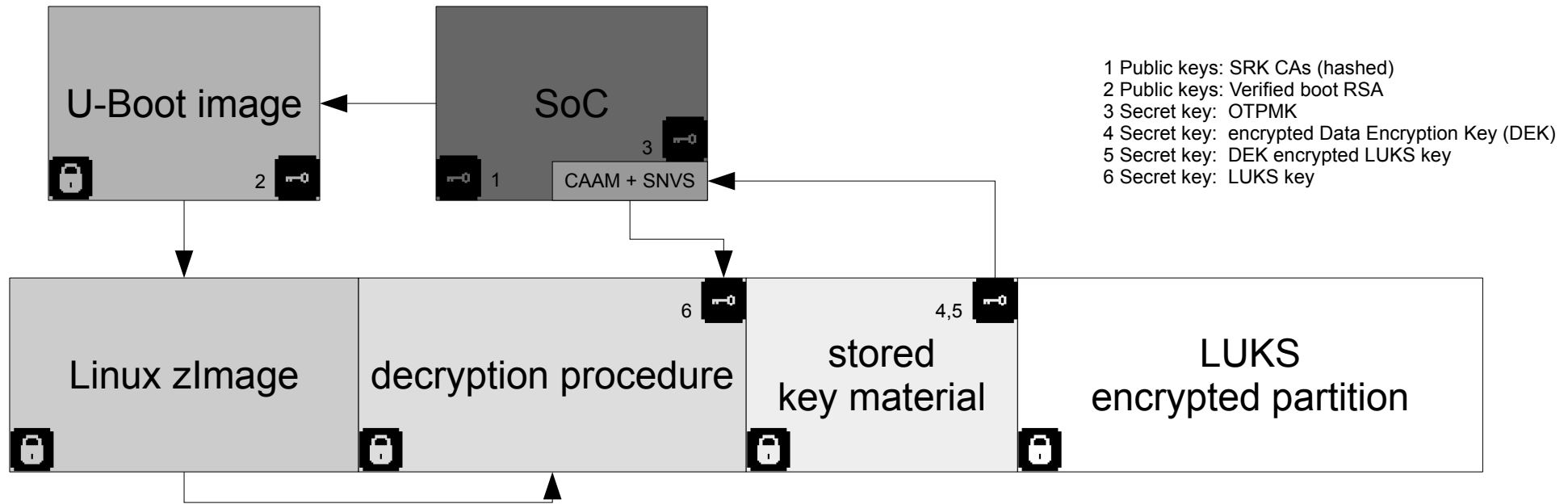
- A random 256-bit blob encryption key (BK) is generated.

- The **blob encryption key is used to encrypt the desired data** via the CAAM AES-CCM function, providing **confidentiality and integrity protection**.

- The blob encryption key is AES-ECB encrypted with a key derived from the OTPMK (BKEK), using a Single-step Key-Derivation Function, resulting in the blob.

- The HAB secure boot sequence, or runtime environment, can directly support authenticated decryption of arbitrary data blobs (including the bootloader image).

# Full chain of trust example - i.MX6UL



- SoC authenticates U-Boot
- U-Boot authenticates Linux
- Linux uses SVNS decrypted key material to unlock the encrypted partition

 key material

 authenticated + encrypted

## Hash generation

```
$ usbarmory_srkttool -h
Usage: usbarmory_srkttool [OPTIONS]
  -1 | --key1 <public key path>    SRK public key 1 in PEM format
  -2 | --key2 <public key path>    SRK public key 2 in PEM format
  -3 | --key3 <public key path>    SRK public key 3 in PEM format
  -4 | --key4 <public key path>    SRK public key 4 in PEM format
  -o | --hash <output filename>    Write SRK table hash to file
  -0 | --table <output filename>   Write SRK table to file
```

## Bootloader signing

```
$ usbarmory_csftool -h
Usage: usbarmory_csftool [OPTIONS]
  -A | --csf_key <private key path> CSF private key in PEM format
  -a | --csf_cert <public key path> CSF public key in PEM format
  -B | --img_key <private key path>  IMG private key in PEM format
  -b | --img_cert <public key path>  IMG public key in PEM format
  -I | --table <SRK table path>      Input SRK table (see usbarmory_srkttool -0)
  -x | --index <SRK key index>       Index for SRK key (1-4)
  -i | --image <filename>            Image file w/ IVT header (e.g. u-boot.imx)
  -o | --output <filename>           Write CSF to file
```



# NXP CAAM + SNVS driver (i.MX6UL)



<https://github.com/inversepath/caam-keyblob>

```
$ sudo modprobe caam_keyblob  
caam_keyblob: Trusted State detected
```

Go userspace implementation:

```
$ caam_tool enc dek.bin dek_blob.bin  
caam_tool: encrypting 32 bytes from dek.bin  
caam_tool: caam_kb_data &{Text:0x49c000 TextLen:32 Blob:0x4a0000 BlobLen:80 Keymod:0x48c010 KeymodLen:16}  
caam_tool: encrypted 80 bytes to dek.bin  
  
$ caam_tool dec dek.bin dek_blob.bin  
caam_tool: decrypting 80 bytes from dek_blob.bin  
caam_tool: caam_kb_data &{Text:0x478000 TextLen:32 Blob:0x474000 BlobLen:80 Keymod:0x412140 KeymodLen:16}  
caam_tool: decrypted 32 bytes to dek.bin
```

# NXP DCP + SNVS driver (i.MX6ULZ)



<https://github.com/inversepath/mxs-dcp>

```
$ sudo modprobe mxs_dcp  
mxs_dcp: Trusted State detected
```

Go userspace implementation:

```
$ dcp_tool enc dek.bin dek_blob.bin  
dcp_tool: enc dek.bin to dek_blob.bin  
dcp_tool: deriving key, diversifier ab  
dcp_tool: done  
  
$ dcp_tool dec dek.bin dek_blob.bin  
dcp_tool: dec dek_blob.bin to dek.bin  
dcp_tool: deriving key, diversifier ab  
dcp_tool: done
```

# INTERLOCK key derivation with SNVS



CAAM + SNVS (i.MX6UL)

```
$ cat interlock.conf
{"hsm": "caam-keyblob:luks"}
$ interlock -c interlock.conf -o derive:ab
cCS2IXA1sBM0n0eU665uhg==
```

DCP + SNVS (i.MX6ULZ)

```
$ cat interlock.conf
{"hsm": "mxs-dcp:luks"}
$ interlock -c interlock.conf -o derive:ab
RReS5gTYc41h3G58FQ2igA==
```

**<https://github.com/inversepath/interlock>**

With support for NXP DCP, SCC2, CAAM the INTERLOCK file encryption front-end is the single OSS app with the most complete support for NXP SoC security. (DEMO!)

# One-Time-Programmable (OTP) fuses



<https://github.com/inversepath/crucible>



# OTP fuse management with crucible



```
$ crucible -l -m IMX6UL -r 1
```

...

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
LOT_NO_ENC[31:0]																																
31	.....																														00	OCOTP_CFG0
Bank:0 Word:1																																
R: 0x00000004																																
W: 0x00000004																																
31	.....																														00	UNIQUE_ID[31:0]
31	.....																														00	UNIQUE_ID
31	.....																														00	SJC_CHALLENGE
31	.....																														00	LOT_NO_ENC[31:0]
31	.....																														00	LOT_NO_ENC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00																	
DIE-X-CORDINATE								DIE-Y-CORDINATE								WAFER_NO				LOT_NO_ENC[42:32]																												
31	.....																														00	OCOTP_CFG1																
Bank:0 Word:2																																																
R: 0x00000008																																																
W: 0x00000008																																																
31	.....																														00	UNIQUE_ID[63:32]																
31	..... 24																														DIE-X-CORDINATE																	
																								23	..... 16								DIE-Y-CORDINATE															
																15	..... 11				WAFER_NO																											
																				10	.....												00	LOT_NO_ENC[42:32]														

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00												
												TAMPE		SI_REV																													
																				21	20	.....												OCOTP_CFG2									
																						19	..... 16								Bank:0 Word:3												
R: 0x0000000c																																											
W: 0x0000000c																																											
TAMPER_PIN_DISABLE																																											
SI_REV																																											

...

# crucible - i.MX6UL example



## Blow a fuse

```
$ sudo crucible -m IMX6UL -r 1 -b 16 blow MAC1_ADDR 0x001f7b1007e3
IMX6UL ref:1 op:blow addr:0x88 off:0 len:48 val:0xe307107b1f000000
```

## Read a fuse

```
$ sudo crucible -m IMX6UL -r 1 -b 16 read MAC1_ADDR
IMX6UL ref:1 op:read addr:0x88 off:0 len:48 val:0x001f7b1007e3
```

## Read a fuse (minimal output for batch operations)

```
$ sudo crucible -s -m IMX6UL -r 1 -b 16 read MAC1_ADDR
001f7b1007e3
```

registers:

OCOTP\_LOCK:

bank: 0

word: 0

fuses:

TESTER\_LOCK:

offset: 0

len: 2

BOOT\_CFG\_LOCK:

offset: 2

Len: 2

...

OCOTP\_MAC0:

bank: 4

word: 2

fuses:

MAC1\_ADDR:

offset: 0

len: 48

OCOTP\_MAC1:

bank: 4

word: 3

OCOTP\_MAC:

bank: 4

word: 4

# crucible - i.MX6UL lock down



It is vital to reduce the low-level SoC attack surface as much as possible and ensure lock down of all relevant fuses.

```
# set device in Closed Configuration (IMX6ULRM Table 8-2, p245)
crucible -m IMX6UL -r 1 -b 2 -e big blow SEC_CONFIG 0b11

# disable NXP reserved mode (IMX6ULRM 8.2.6, p244)
crucible -m IMX6UL -r 1 -b 2 -e big blow DIR_BT_DIS 1

# Disable debugging features (IMX6ULRM Table 5-9, p216)
# * disable Secure JTAG controller
# * disable JTAG debug mode
# * disable HAB ability to enable JTAG
# * disable tracing
crucible -m IMX6UL -r 1 -b 2 -e big blow SJC_DISABLE 1
crucible -m IMX6UL -r 1 -b 2 -e big blow JTAG_SMODE 0b11
crucible -m IMX6UL -r 1 -b 2 -e big blow JTAG_HE0 1
crucible -m IMX6UL -r 1 -b 2 -e big blow KTE 1

# To further reduce the attack surface:
# * disable Serial Download Protocol (SDP) READ_REGISTER command (IMX6ULRM 8.9.3, p310)
# * disable SDP over UART (IMX6ULRM 8.9, p305)
crucible -m IMX6UL -r 1 -b 2 -e big blow SDP_READ_DISABLE 1
crucible -m IMX6UL -r 1 -b 2 -e big blow UART_SERIAL_DOWNLOAD_DISABLE 1
```

[https://github.com/inversepath/usbarmory/wiki/Secure-boot-\(Mk-II\)](https://github.com/inversepath/usbarmory/wiki/Secure-boot-(Mk-II))



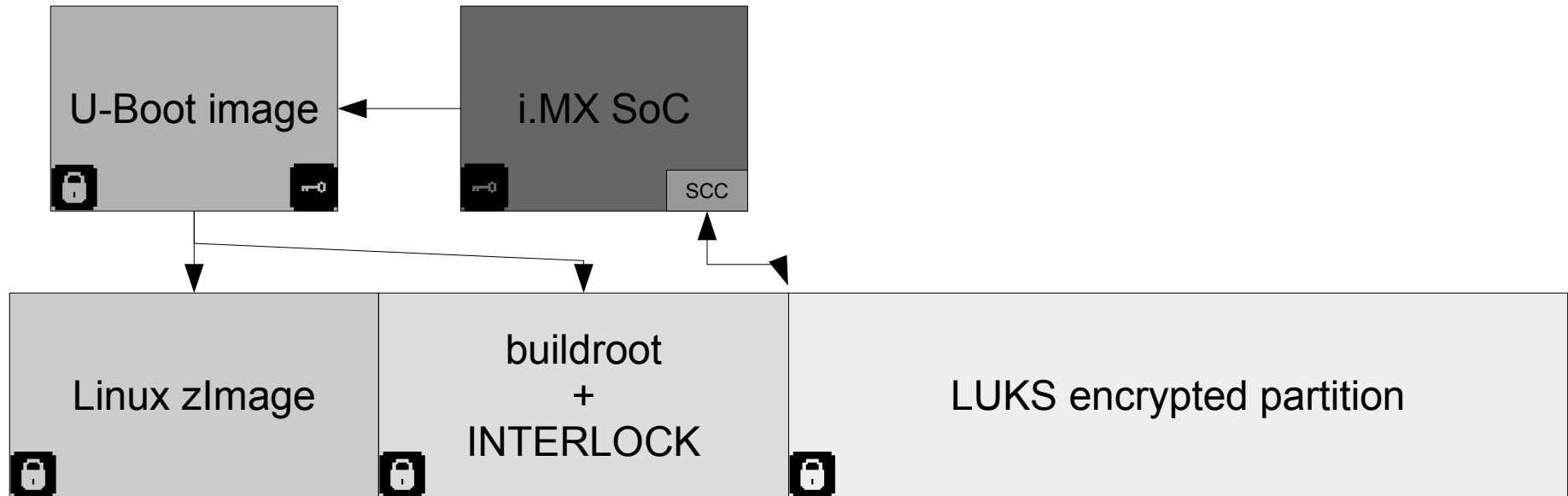
# Buildroot



<https://github.com/inversepath/usarmory/tree/master/software/buildroot>

Custom buildroot profiles allow compilation of bootloader, kernel, runtime environment and target application with an automatic cross-compilation process.

```
make BR2_EXTERNAL=${USBARMORY_GIT}/software/buildroot interlock_mark_one_defconfig  
make BR2_EXTERNAL=${USBARMORY_GIT}/software/buildroot # yes, it's that easy!
```



## U-BLOX ANNA-B112



The addition of a Bluetooth module opens up a variety of new use cases.

The trust towards the host can now be limited to the strict amount necessary.

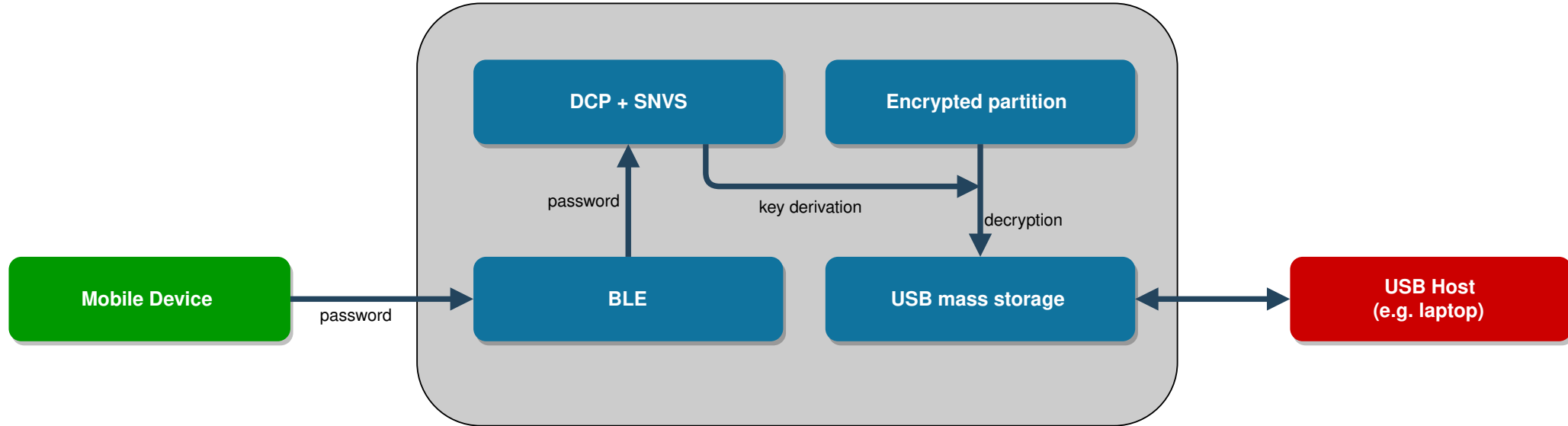
Out-of-band interaction with a mobile app is now possible

The ANNA-B112 module supports an "OpenCPU" option to allow arbitrary firmware, replacing the built-in u-blox one, on its Nordic Semiconductor nRF52832 SoC.

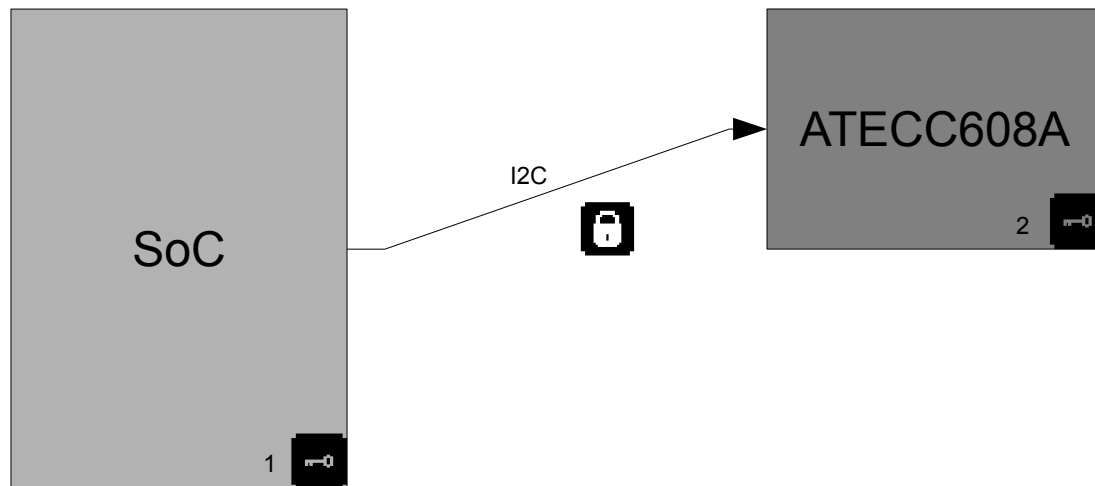
This allows provisioning of the SoC with Nordic SDK, Wirepas mesh, ARM Mbed or arbitrary user firmware. The nRF52832 SoC features an ARM Cortex-M4 CPU with 512 kB of internal Flash and 64 kB of RAM.

DEMO

# Full chain of trust example - i.MX6UL



# Rollback protection + external keyring



1 Secret keys: ReadKey, WriteKey

2 Slots (16x) for key, certificates or data  
High Endurance Monotonic Counters (2x)  
OTPs (512-bit)

 key material

 authenticated + encrypted

The SoC can establish a secure session with the ATECC608A, using safely stored read and write keys, certificates or data.

This allows secure key/certificate access or use, additionally two High Endurance Monotonic Counters can be used for rollback protection.

Provides an additional hardware keyring for (partial) mitigation of further HAB issues.

## i.MX6UL - Security audit



Against Silicon Revision 1.2 and HAB 4.1 or greater, meaning P/Ns “AB” or greater, with patched HABv4.

Completed as an internal + third party security audit for HABv4 as well as our buildroot chain of trust implementation.

### Conclusions

No further issues have been identified in the patched boot ROM.

Freescall kernel module issues (invalid error values, NULL pointer exceptions, various operational errors), all resolved in our own caam-keyblob driver implementation.

Several U-Boot issues identified and resolved (CVE-2018-18439, CVE-2018-18440)

# USB armory Mk II – public repositories



Documentation

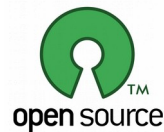
**<https://github.com/inversepath/usbarmory/wiki>**

caam-keyblob (i.MX6UL) - CAAM + SNVS driver

**<https://github.com/inversepath/caam-keyblob>**

mxs-dcp (i.MX6ULL) - DCP + SNVS driver

**<https://github.com/inversepath/mxs-dcp>**



INTERLOCK with CAAM/DCP support

**<https://github.com/inversepath/interlock>**

crucible - OTP fusing tool

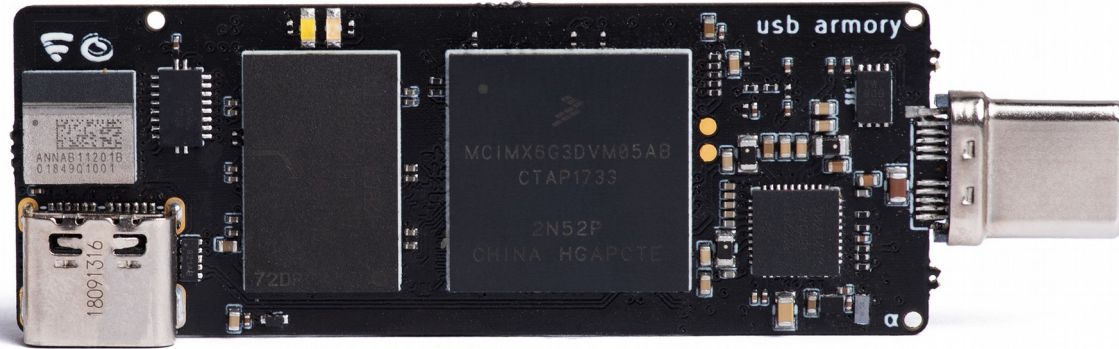
**<https://github.com/inversepath/crucible>**

Buildroot profile for embedded INTERLOCK distribution

**<https://github.com/inversepath/usbarmory/tree/master/software/buildroot>**

**Thank you!**

Questions?



**Andrea Barisani**

Head of Hardware Security  
andrea.barisani@f-secure.com

**<https://github.com/inversepath/usbarmory>**