# Practical reverse engineering of ECC-based authentication device with zero knowledge

Dr Sergei Skorobogatov

*http://www.cst.cam.ac.uk/~sps32* *email: sps32@cam.ac.uk*

In collaboration with Dr Markus Kuhn and Shih-Chun You

UNIVERSITY OF CAMBRIDGE

Dept of Computer Science and Technology

# Outline

- Introduction: challenges to prevent counterfeiting
- Authentication devices
- Security features and challenges
- Reverse engineering of proprietary authentication device from scratch
- Analysis of vulnerabilities
- Reverse engineering milestones
  - communication and authentication
  - memory access
  - special function registers
  - searching for backdoors and Trojans
  - bypassing memory encryption
- Cloning and its limitations
- Future work
- Conclusion

# Introduction

- Senior Research Associate at the University of Cambridge
  - Hardware Security research (attack technologies) since 1995
  - test microcontrollers, smartcards, FPGAs and SoCs for security
  - knowledge: chemistry, electronics, physics (MSc), computer science (PhD)
- Research interests
  - finding real solutions to “impossible problems”
  - revisiting forgotten techniques
  - developing new attack methods
  - testing challenging hardware devices for vulnerabilities
- Counterfeit prevention based on authentication is important
  - was involved in analysis of authentication devices since 1996
  - car industry relied on authentication for security purposes since 1980s
  - consumables often use strong authentication since 1990s
    - memory cartridges (servers and game consoles)
    - printer cartridges
    - batteries
  - medical equipment, life-critical and safety-critical systems

# Evolution of authentication: 1980s…today

- Security via obscurity in old days – until 1990s
  - very simple solutions based on serial numbers (DS2401 – serial ID chip)
  - devices with proprietary communication protocols or no protocol at all
  - **Attack methods:** eavesdropping or brute forcing
- Challenging hardware security – early 2000s
  - security via obscurity (weak proprietary encryption)
  - devices based on symmetric cryptography (DES, AES)
  - authentication using hash functions (DS2432 – SHA-1 chip)
  - **Attack methods:** reverse engineering, firmware extraction, memory probing
- Advanced hardware security – 2010s
  - countermeasures against side-channel attacks and glitching
  - countermeasures against physical attacks (sensors, memory encryption)
  - devices with advanced fabrication process: 45nm to 90nm, 5–7 metal layers
  - authentication using asymmetric cryptography (RSA, ECC)
  - **Attack methods:** reverse engineering, chip modification, data bus probing

# Symmetric vs Asymmetric authentication

- Symmetric authentication
  - Host requests information (ID, user data)
  - Host sends random number
  - Device computes Hash and sends it back
  - Host generates derived key and computes Hash
  - device is authentic if both hashes are the same
- Asymmetric authentication
  - Host requests information (ID, user data, Public key, Certificate)
  - Host checks if the Public key has a valid certificate
  - Host sends random challenge $r \cdot G$ ($r$ – random number, $G$ – base point)
  - Device computes $q \cdot (r \cdot G)$ and sends it back ($q$ – private key)
  - Host computes $r \cdot Q$ ($Q = q \cdot G$, $Q$ – public key)
  - device is authentic if both numbers match: $r \cdot (q \cdot G) = q \cdot (r \cdot G)$

# Symmetric vs Asymmetric authentication

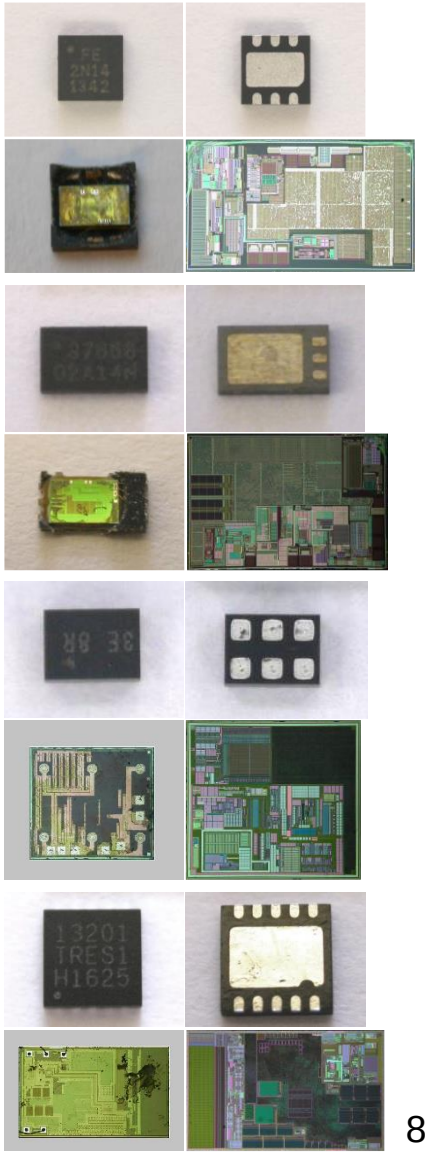
- Symmetric authentication
  - each device stores unique key, hence, not big deal if compromised
  - Host stores everything needed for producing cloned devices
  - Key derivation could be based on strong cryptography
  - if devices have weak security an attacker could extract large set of keys
  - algorithm could be implemented on simple devices
- Asymmetric authentication
  - each device stores unique key, hence, not big deal if compromised
  - Host does not store any key – only algorithm to verify validity of the secret key
  - if devices have weak security an attacker could extract large set of keys
  - algorithm requires devices with advanced computing power or with crypto-engine
- Aim of an attacker: bypass authentication without being detected
  - ideally: be able to generate unique device ID, secret key and signatures
  - realistically: be able to extract thousands of real IDs + secret keys + signatures
  - real world applications: make sure the solution is adequately secure

# ECC-based authentication devices

- Texas Instruments: BQ40Z80
  - devices with documentation and evaluation/development kits are available
- Maxim Semiconductors: DS28C36, DS28E36, DS28E38
  - devices and evaluation kits with documentation are available
  - datasheets and libraries can be found
- Microchip(former Atmel): ATECC508A, ATECC608A
  - devices with some documentation are available, restricted development kits
- Infineon: SLE95050, SLE95200, SLE95250, SLS32AIA
  - devices can be found, but abridged datasheets with very little information
  - limited availability of evaluation kits, restricted development kits
- NXP: A1006, A1007, A7101, A7102
  - devices are available, but abridged datasheets with very little information
  - restricted development kits
- ST Microelectronics: ATSAFE-A100
  - devices and tools not available: based on real smartcard chip (EAL5+ certified)

# Infineon ECC-based authentication devices

- SLE95050: Origa 1
  - WQFN-6 package: 2.5mm×2.5mm×0.75mm
  - die marking: INFINEON 2010 M0620A31 IFAP DC IMM
  - ~180nm fabrication process, no sensor mesh, ECC engine
  - 0.2kB EEPROM, full logic design
- SLE95200: Origa 2
  - USON-3 package: 3.3mm×1.5mm×0.6mm
  - ~180nm fabrication process, no sensor mesh, 163-bit ECC
  - 0.8kB EEPROM, MaskROM, CPU core
- SLE95250: Optiga Trust B
  - TSNP-6 package: 1.5mm×1.1mm×0.4mm
  - ~90nm fabrication process, no sensor mesh, 131-bit ECC
  - 0.6kB EEPROM, full logic design
- SLS32AIA: Optiga Trust E
  - USON-10 package: 3mm×3mm×0.6mm
  - ~90nm fabrication process, no sensor mesh, 256-bit ECC
  - 300kB Flash, MaskROM, SRAM, CPU core

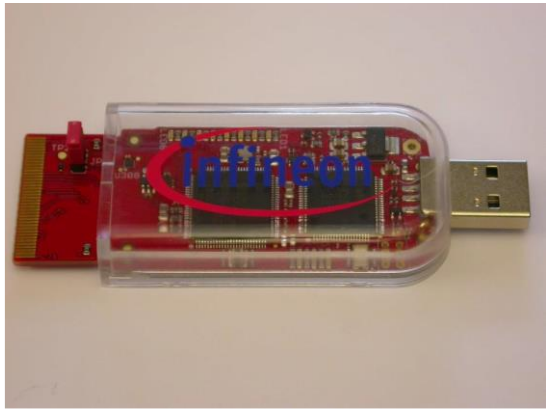


# Infineon Optiga Trust B (SLE95250)

- Devices are available from distributors
- Evaluation Kit is available from distributors
- Publicly available datasheet contains very limited information
  - package, pinout, connection, power supply
  - communication interface is SWI (single wire), but no information on it at all
  - modes of operation without any details, no details on 512-bit user NVM
  - 131-bit ECC engine, 163 bits certificate (ODC)
- No information about
  - SWI interface (waveforms, bit encoding etc.)
  - communication protocol and commands
  - NVM reading and writing
  - usage of Life Span counter
  - ODC signature verification process
  - ECC curve parameters and authentication
  - MAC function used in authentication

# Optiga Trust B Evaluation Kit

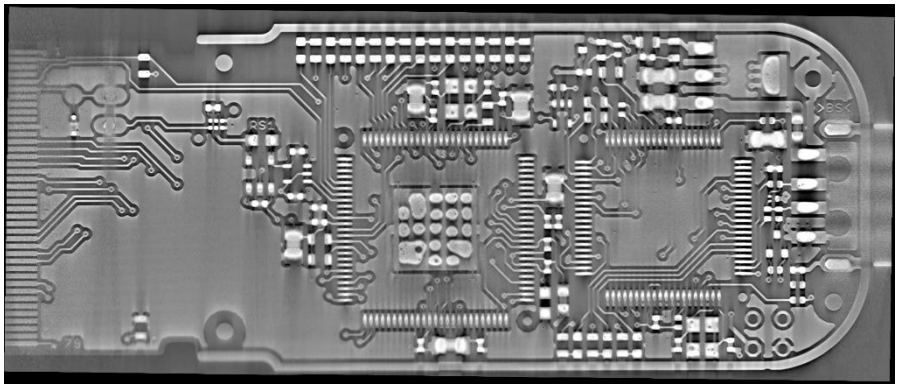
- Windows GUI that shows authentication steps without details
- User guide has only information about GUI usage
- No schematic or firmware provided with the Kit
- Evaluation Kit could give a lot of clues
  - logic analyser shows SWI communication waveforms
  - USB traffic can be monitored using PC tools
- Internet search revealed that SWI is based on MIPI BIF standard
  - Infineon patent (US7636806) describes the interface and communication
  - Infineon IEC62700 proposal describes data encoding and transactions
- We can start talking to the chip via SWI interface

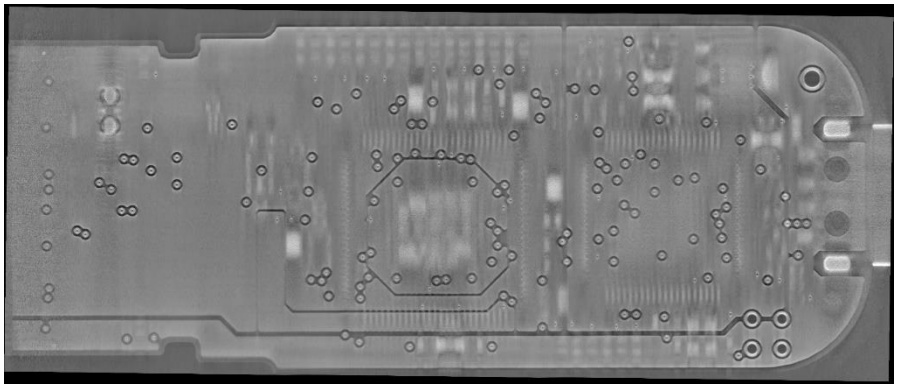


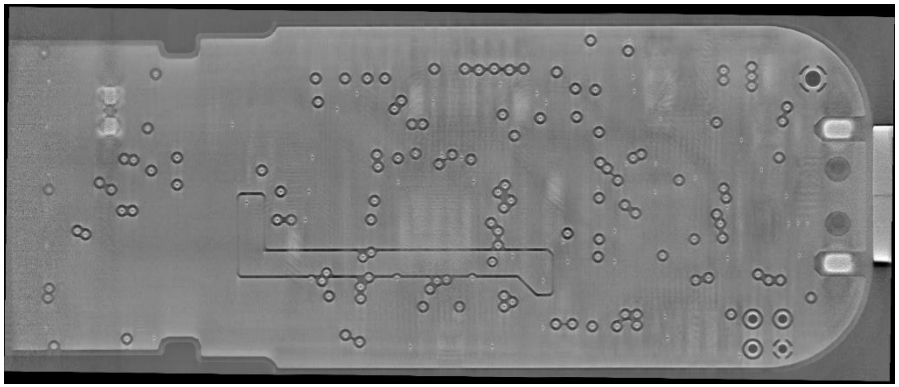


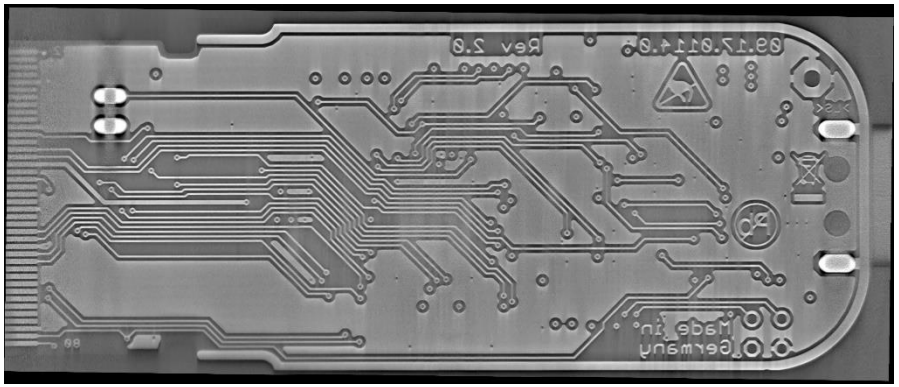
# Reverse engineering of the Evaluation Kit

- Circuit diagram of the Evaluation Kit (4-layer PCB)
  - took high-resolution photos of top and bottom sides
  - performed identification of all components
  - obtained 3D X-ray images of the PCB with components (alternative to polishing)



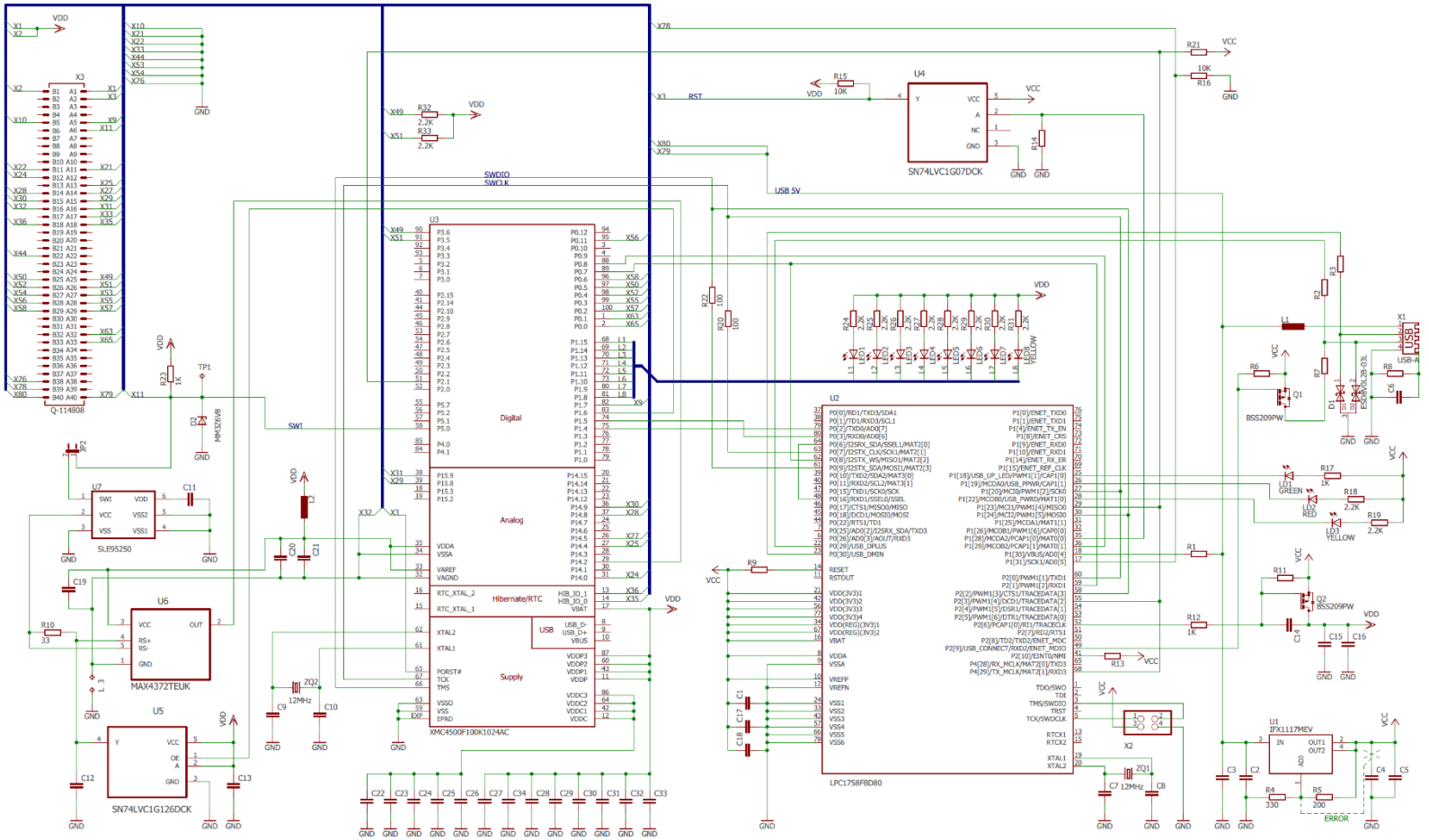






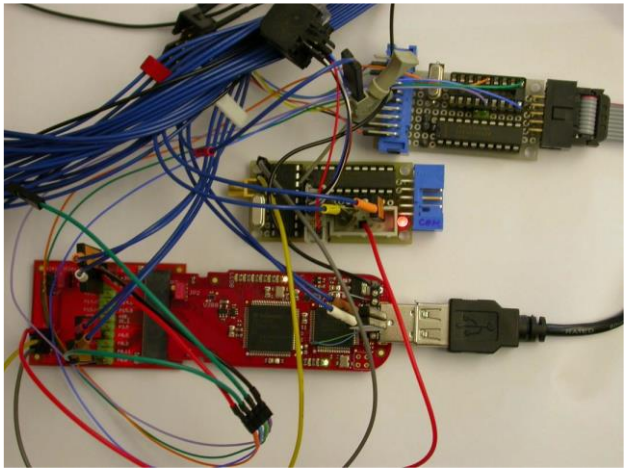
# Reverse engineering of the Evaluation Kit

- Circuit diagram created in Autodesk Eagle CAD
  - libraries available for all components except SLE95250 (custom drawing)
  - XMC4500 controls SLE95250, LPC1758 communicates with USB



# Reverse engineering of the Evaluation Kit

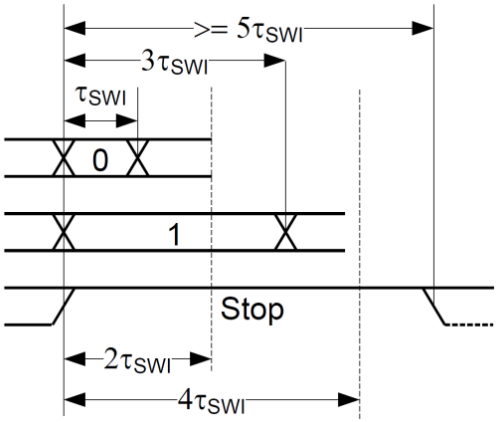
- Based on Infineon XMC4500 Cortex M4 microcontroller
- Logic analyser reveals hidden debug port
  - Port P0.1 is configured as UART and present on daughter board
  - debug information sent in parallel to SWI communication
- Another ARM microcontroller is used as USB bridge
  - talks via UART with XMC4500 (P1.4 and P1.5) and sends/receives data from PC

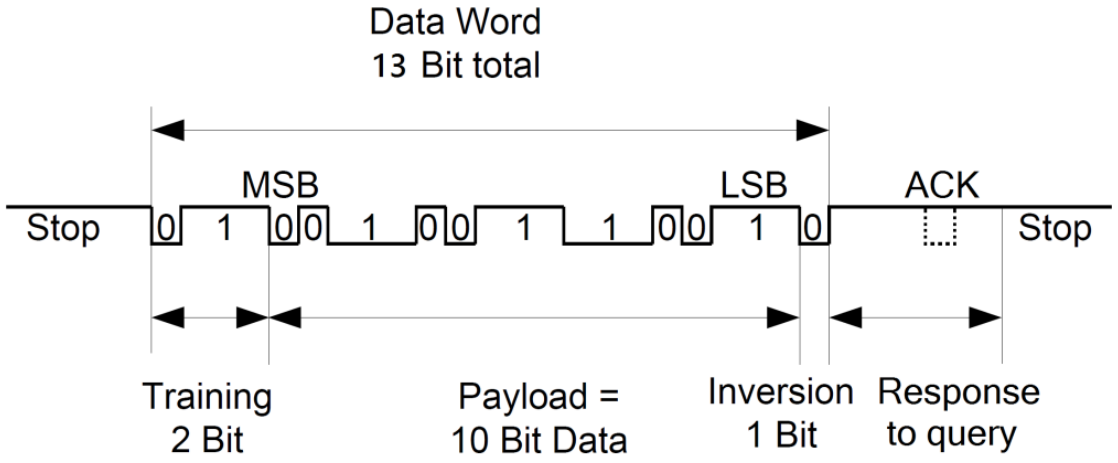


```
********************************************************************************
Infineon Technologies Origa1&2 Console Test Programs
Code Compiled: Mar  3 2017 15:04:50
Code Version: 3.2.0
RTC: 0:0:0
Note: Using TeraTerm, change the setting at Setup->Terminal->New-line.>Transmit:CR+LF
********************************************************************************
Entered GUI mode...
Host Configuration: Baud Rate=10KHz-Tau-50uS
Waiting for GUI command...

COM_DETECT_UNIQUE_ID
COM_DETECT_UNIQUE_ID: SWI Interface
Power Cycle completes.

UID Found: 1
COM_GET_UNIQUE_ID
COM_PWR_TRAIN.
COM_SELECT_ORIGA: Device currentSelectedUID 0
Enumerate O1.
```

# SWI communication

- Power up and idle state is ‘HIGH’
- Power down is ‘LOW’
- Logic ‘0’ is short pulse ($\tau_{SWI}$), ‘1’ is long pulse ($3\tau_{SWI}$), STOP is $>5\tau_{SWI}$
- Acknowledge by presence of ‘0’ within time window
- No need for stable clock, variable speed (training bits), $\tau_{SWI}$=1…50µs




Source: Infineon IEC 62700 proposal

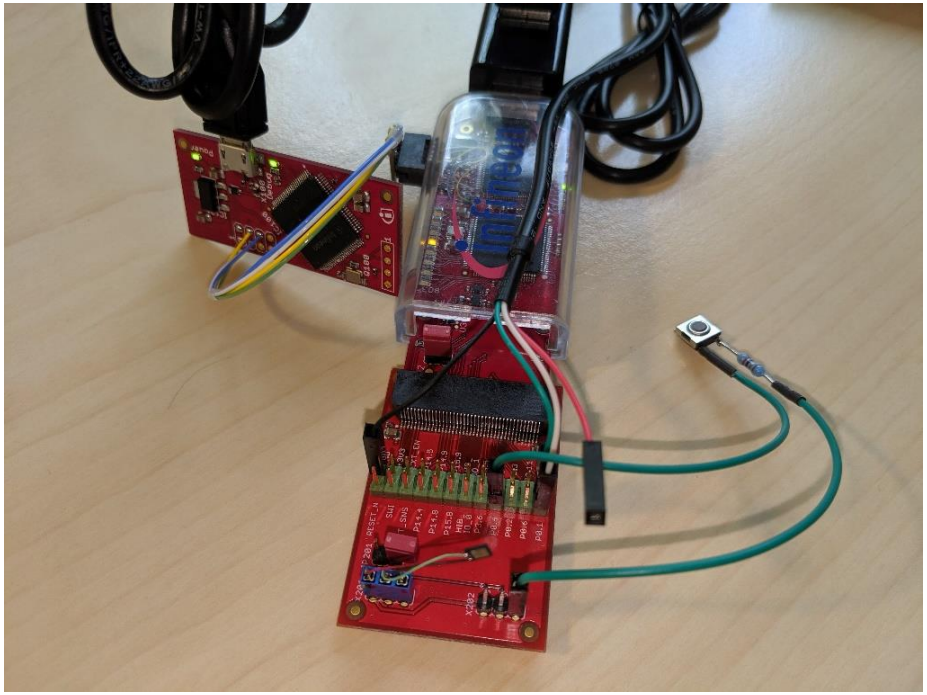
# Reverse engineering of the Evaluation Kit

- Logic analyser helps in aligning SWI, debug and USB data

| Event | Commands from USB chip (UART) | Data to USB chip (UART) | SWI bus 15-BCF,14-!BCF,9..0-Data | Debug terminal |
|---|---|---|---|---|
| Start GUI | 00 06 01 01 01 01 00 00<br>00 00 00 80 60 FD 29 73<br>04 00 00 00 2A 00 2A 00<br>2A 00 20 00 00 00 00 00<br>00 00 00 80 60 FD 29 73<br>03 00 00 00 2A 00 2A 00<br>2A 00 00 00 00 00 00 80<br>60 FD 29 73 07 00 00 00 | | | COM_DETECT_UNIQUE_ID<br>COM_DETECT_UNIQUE_ID: SWI Interface |
| SWI init | | | SWI=0 for 1.8 ms<br>SWI=1 for 9.0 ms<br>8000 | Power Cycle completes. |
| Select ID | 00 06 03 01 00 02 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00 | | | COM_SELECT_ORIGA: Device currentSelectedUID 0 |
| Enumerate | | 03 05 03 00 03 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 FF | 8034<br>8030<br>8034 8034 8034 8035<br>8035 8034 8034 8034<br>8034 8034 8035 8034<br>8035 8034 8035 8034<br>8034 8034 8034 8034<br>8034 8035 8035 8035<br>8034 8034 8035 8034<br>8034 8034 8034 8034<br>8034 8034 8035 8035<br>8035 8034 8035 8035<br>8034 8035 8035 8034<br>8034 8035 8035 8034<br>8034 8034 8034 8034<br>8034 8034 8034 8034<br>8035 8034 8034 8034<br>8034 8034 8035 8034<br>8035 8034 8035 8035<br>8034 8034 8034 8035<br>8034 8034 8035 8035<br>8034 8034 8034 8035<br>8034 8034 8035 8034<br>8034 8035 8035 8034<br>8035 8034 8034 8035<br>8020 8050 4100 4220 4001<br>8020 8050 4100 4221 4000<br>8100 8201 | Enumerate O1.<br><br>IFX OPTIGA Trust B Engineering Sample.<br>ul_Low 691233B0.<br>ul_High 8200663B.<br><br>Engineering OPTIGA Trust B found.<br><br>OPTIGA Trust B SLE95250. |
| Get Verif | 00 05 2C 00 2C 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00 | 0F 0A 2C 05 42 8C EE 98<br>01 90 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00<br>00 00 00 00 00 00 00 00 | | SWI: Verify ECCE Response Value.<br><br>OUL Provides Z&X-Response:<br>OUL ZResponse (131-bit):<br>[6f6099f6, 967c2c6b, 815d400c, 80fa78b7, 00000000]<br><br>OUL XResponse (131-bit):<br>[1e1c7ccd, 51bf8517, 00000000, 00000000, 00000000]<br><br>OPTIGA Trust B CheckValue (131-bit):<br>{0x7a52, 0xe042, 0x9c0d, 0x1a83, 0xc266, 0x3e6c, 0x1f1c, 0x0105, 0x0005 }<br><br>OPTIGA Trust B Public Key (131-bit):<br>{0xa4f4e3c5, 0x1db1e2b3, 0xaf619148, 0x575037f4, 0x0006 }<br>MAC64 Computation:<br>Public Key: [0xa4f4e3c5 0x1db1e2b3 0xaf619148 0x575037f4 0x6]<br>CheckValue: [0xe0427a52 0x1a839c0d 0x3e6cc266 0x1051f1c 0x5]<br>Z_result: [0x6f6099f6 0x967c2c6b 0x815d400c 0x80fa78b7 0x0 ] |

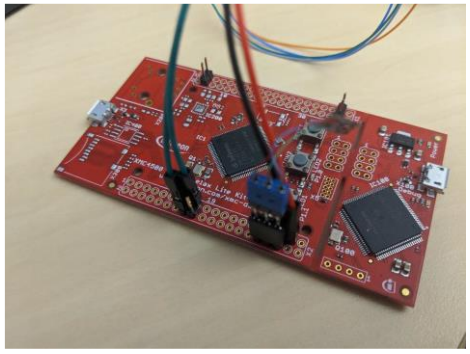
# Reverse engineering of the Evaluation Kit

- Debug port of XMC4500 wired only to LPC1758
  - can be traced on the PCB using the circuit diagram and wired to connector
  - J-Link JTAG debugger controller used with OpenOCD and Ozone J-Link debugger
    - CPU Run/Hold control with 6 breakpoints
    - Full Memory access and Flash programming
  - Code compilation using GCC or DAVE



# Firmware decompilation

- Windows GUI program does not do any verification
  - possible to turn it back into C# using .NET decompiler: reveals names of functions
- XMC4500 performs the ECC authentication as a host then talks to PC
- Firmware was extracted with J-Link debugger
- Decompilation using Ghidra decompiler tool
  - understanding of all operations and commands
  - understanding SWI subroutines and ECC authentication flow
- SWI communication was re-implemented on XMC4500 Relax Lite Kit
- ECC authentication was implemented in Python
- Turned into successful practical course for Master students at CAM



```
?>v
Execute function: Power up VCC line and Power cycle SWI
devices

?>d
Execute function: Detect SWI devices
Found SWI devices: 1
Found SWI Device ID: H:C410023C L:080E2298 V:2A18 P:2007

?>p
Execute function: Get ODC and Public Key from selected
Device
Device ODC: 25 20 9D E0 CA 96 62 A3 2C AD F2 A3 53 7C A8
72 F6 95 6F EF D8 CE 6E EE F3 56 AF 01 43 ED A5 CF 43 5D
CA B1 77 16 DB 7E A6 BD 0A 7F 51 A6 E1 66
Device Public Key: CB 29 05 74 A5 8D 3D C4 9D 0A 27 3E
82 67 A8 54 AF 1F
```

```
?>n
Execute function: Read NVM from selected Device

0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 A0 86 01 00 00 00 00 00
0050: 04 00 00 00 00 00 92 16 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0: 00 00 00 00 00 00 00 00 25 20 9D E0 CA 96 62 A3
00C0: 2C AD F2 A3 53 7C A8 72 F6 95 6F EF D8 CE 6E EE
00D0: F3 56 AF 01 43 ED A5 CF 43 5D CA B1 77 16 DB 7E
00E0: A6 BD 0A 7F 51 A6 E1 66 CB 29 05 74 A5 8D 3D C4
00F0: 9D 0A 27 3E 82 67 A8 54 AF 1F F2 16 E7 40 D9 58
```

# SWI commands

- Master commands

| | |
|---|---|
| 800 | Reset |
| 9xx, Ayy | Select device with global address XX:YY |
| 810 | Status query |
| 830, 83x, 83y | Query bit (x=6,7), y=4/5 (select bit) |
| 830, 83x | Select bit (x=4/5) |

- Volatile Memory access

| | |
|---|---|
| 82n, 850, 5xx, 7yy, 7zz/7zz | Read 1/2/4/8 bytes (ZZ) at address XX:YY |
| 82n, 850, 5xx, 6yy, 7zz/7zz | Write 1/2/4/8 bytes (ZZ) to address XX:YY |

- Registers access

| | |
|---|---|
| 82n, 851, 5xx, 7yy, 7zz/7zz | Read 1/2/4/8 bytes (ZZ) at address XX:YY |
| 82n, 851, 5xx, 6yy, 7zz/7zz | Write 1/2/4/8 bytes (ZZ) to address XX:YY |

- Special commands

| | |
|---|---|
| 8C1 | Start ECC computation |
| 810 | Query device status (ACK – ready) |

# SWI registers

- Data Buffers

  [0010 – 0017] ECC result, value X

  [0010 – 001F] NVM read buffer

  [0020 – 002F] NVM write buffer

  [0030 – 003F, 0330] ECC result, value Z

  [0040 – 004F, 0340] ECC challenge

- NVM access

  [0274] NVM control (set address, select buffer, read/write, start [WR]/status[RD])

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 – ready<br>1 – start | 0 – read<br>1 – write | select buffer | NVM address [7:3] | | | | |

  [0272] NVM command

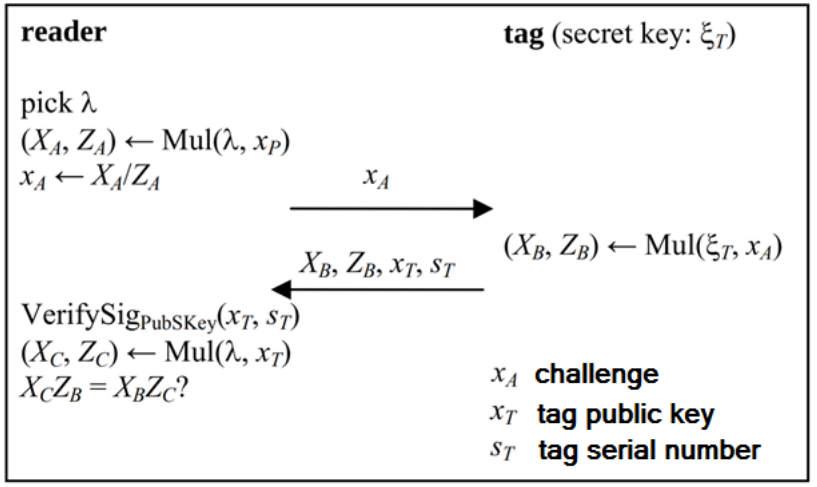
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ? | ? | 0 – direct<br>1 – count | length, bytes: 00 – 1,<br>01 – 2, 10 – 4, 11 – 8 | | NVM address [2:0] | | |

# NVM access

- NVM read sequence
  - 820, 851, 502, 674, 4xx — XX is Addr[2:0]
  - 820, 851, 502, 672, 4xx — XX is 0x80+Addr[7:3]
  - 820, 851, 5xx, 7yy, 7zz/7zz — YY:XX address of NVM read buffer, ZZ is data
- NVM write sequence
  - 820, 851, 5xx, 6yy, 4zz/4zz — YY:XX address of NVM write buffer, ZZ is data
  - 820, 851, 502, 674, 4xx — XX is Addr[2:0]
  - 820, 851, 502, 672, 4xx — XX is 0xC0+Addr[7:3]
  - 820, 851, 502, 672, 7xx — XX bit 7 is status (0 – ready)
- Life Span counter decrement
  - 820, 851, 502, 674, 420 — select COUNTER mode
  - 820, 851, 502, 672, 489 — decrement COUNTER
  - 820, 851, 502, 672, 7xx — XX bit 7 is status (0 – ready)

# Reverse engineering of Authentication

- Challenge-response protocol
  - based on variant of Braun-Hess-Meyer[1] (2008)
  - optimized for low hardware footprint
  - scalar-point multiplication uses Montgomery method due to López-Dahab (CHES 1999), which does **not** require ***y*** coordinate or curve parameter ***a***
  - uses projective coordinates
    - avoids any division in the authentication chip
    - allows randomization of calculation (for SCA countermeasures)




Michael Braun, Erwin Hess, Bernd Meyer: Using Elliptic Curves on RFID Tags. IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.2, February 2008. Siemens AG, Corporate Technology

# Public key certificate

- 163-bit EC parameters (certificate signature)
  - curve $y^2 + xy = x^3 + ax^2 + b$, coordinates x, y in field $GF(2^{163})$
  - a = 0
  - b_sqrt_coeff = **0x00DBDC938470FBF823DD55C8E85BDCAE9A797268E9**
  - $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ (as in SEC 2 curves sect163{k1,r1,r2}, confirmed by trial)
  - group order n = **0x020000000000000000000072C82CAB1C1899B2CC87**
  - the base point P = (Px,Py): (labelled as "ODC_AFFINE_X" and "ODC_AFFINE_Y", resp.)

    Px = **0x05D52DEDC9F295BF052A645A3DC5A16DABC24277C4**

    Py = **0x07C0EFB574227CD1A22280C10F6CC79DB61F06BA90**
  - the public key T = (Tx,Ty): (labelled as "ODC_TRUST_ROOT")

    Tx = **0x04DE5BCD40DABBA0A517F7414903FB02A84E1C2CF6**

    Ty = **0x0212E3B51C0B1F22F3A35FE797398336B67A549B52**

# ECC authentication

- 131-bit EC parameters (challenge–response)
  - curve $y^2 + xy = x^3 + ax^2 + b$, coordinates x, y in field $GF(2^{131})$
  - a = irrelevant for protocol, due to the Montgomery method used
  - b_sqrt = 0x00E4808F8949D33C69E070A5F82C3633D9
  - $f(x) = x^{131} + x^8 + x^3 + x^2 + 1$ (as in SEC 2 curves sect131r{1,2}, confirmed by trial)
  - group order: not yet determined
  - the base point G = (Gx,Gy):

    Gx = **0x03651A4282AE22C4FC6C20C9B7281EC1F5**

    Gy = irrelevant for protocol, due to the Montgomery method used
  - the public key Q = (Qx,Qy): (of one sample device)

    Qx = **0x06575037F4AF6191481DB1E2B3A4F4E3C5**

    Qy = irrelevant for protocol, due to the Montgomery method used

# Reverse engineering of Authentication

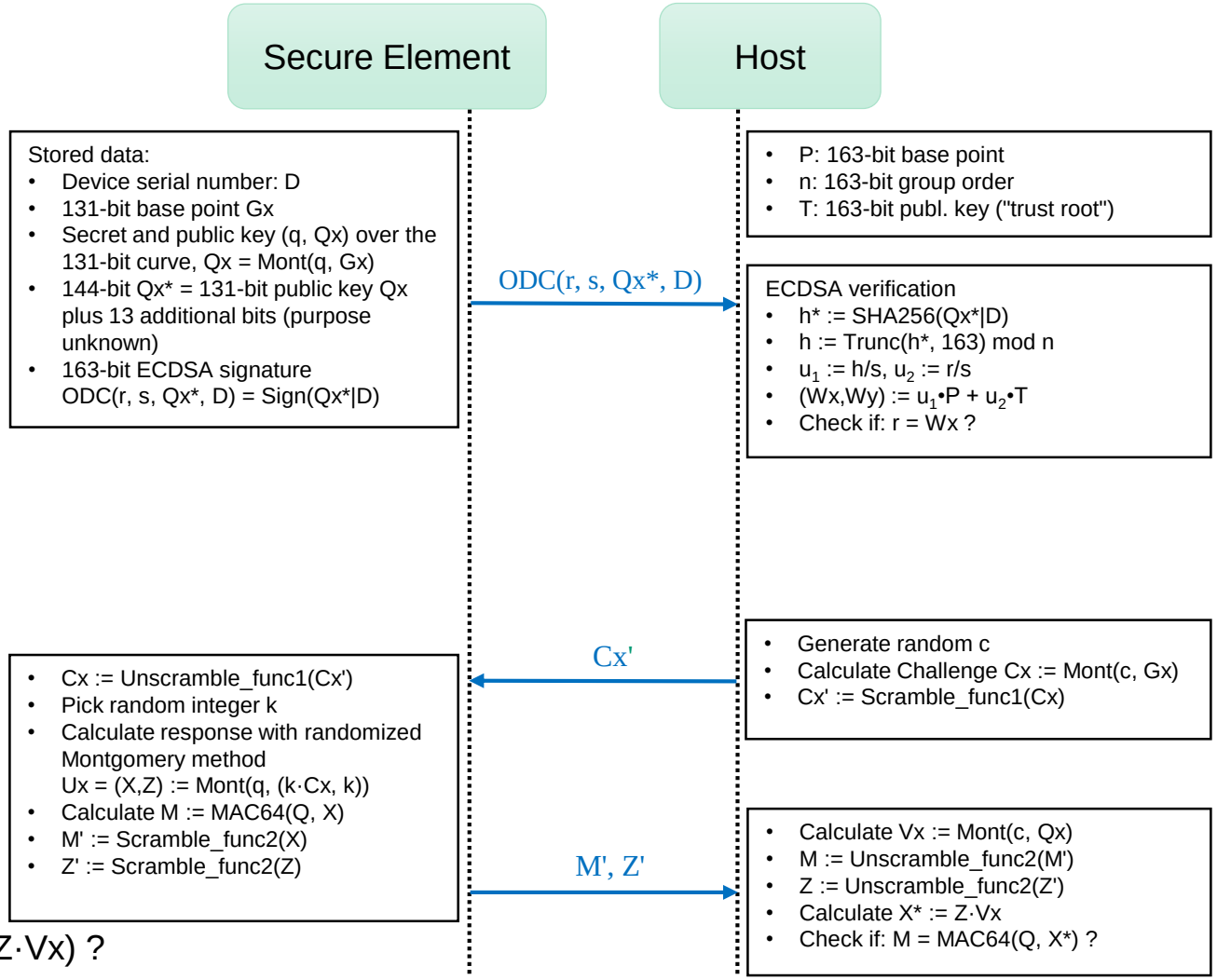
Read certificate ODC, binding public key Q to device serial number D

Verify the ODC signature over the 163-bit curve

Share the secret:
Vx = Mont(c, Qx)
= Mont(q, Cx) = X/Z = Ux

Projective coordinates:
Although result (X, Z) is randomized as (k'·X,k'·Z), quotient Ux = (k'·X)/(k'·Z) = X/Z remains deterministic

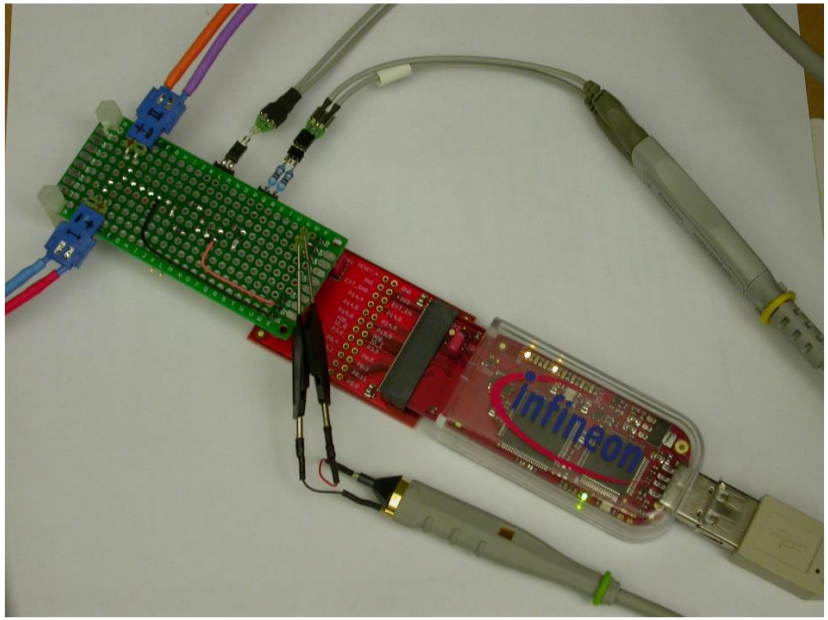
Verify if the secret has been shared successfully:
MAC64(Q, Z·Ux) = MAC64(Q, Z·Vx) ?




Great dot •: scalar muplication over EC ((Wx,Wy) := $u_1$•P + $u_2$•T), both coordinates x, y get involved.
Mont(s, Px): scalar muplication over EC with Montgomery method, only coordinate x gets involved.
Small dot ·: muplication over GF($2^{131}$)/f(x), f(x) = $x^{131} + x^8 + x^3 + x^2 + 1$

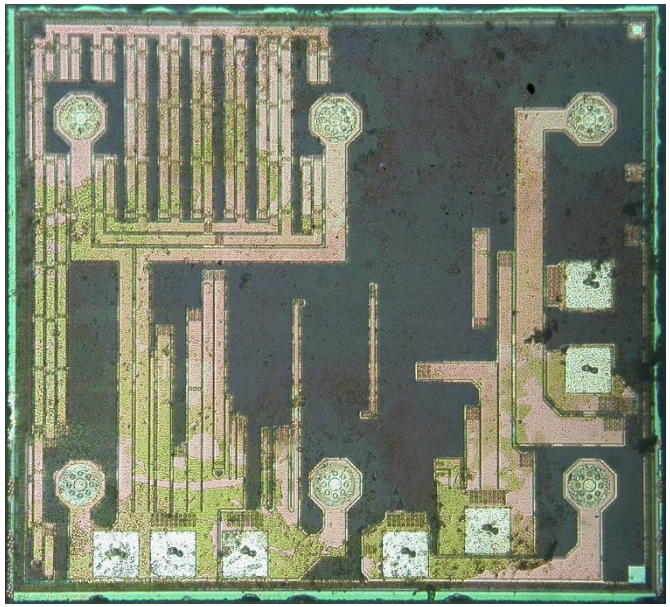
# Side-channel analysis

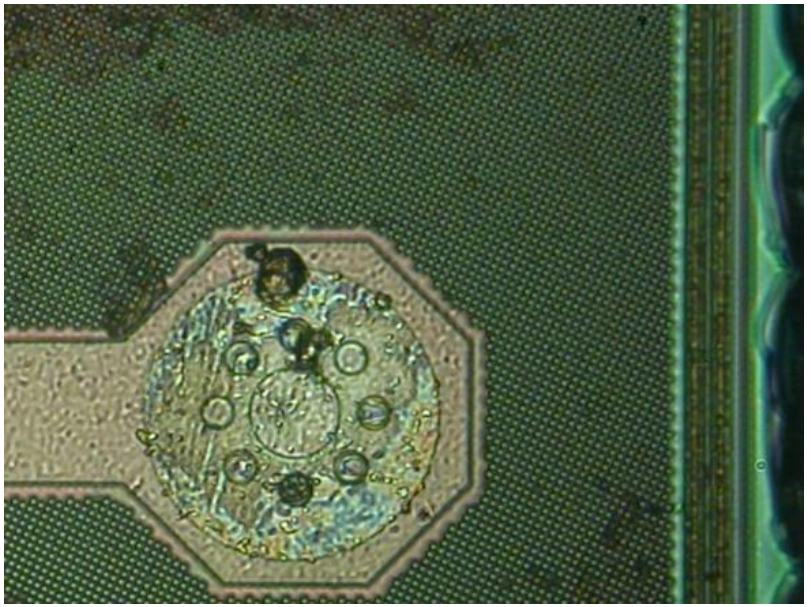
- Dedicated board was built for acquiring power traces
- External power supply through 10Ω resistor and use differential probe
  - high level of noise
  - base frequency of 8MHz with multiple harmonics
- Presence of SCA countermeasures
  - unstable internal RC clock source
  - randomised ECC computation: different response for the same ‘random’ challenge




# Optical fault injection

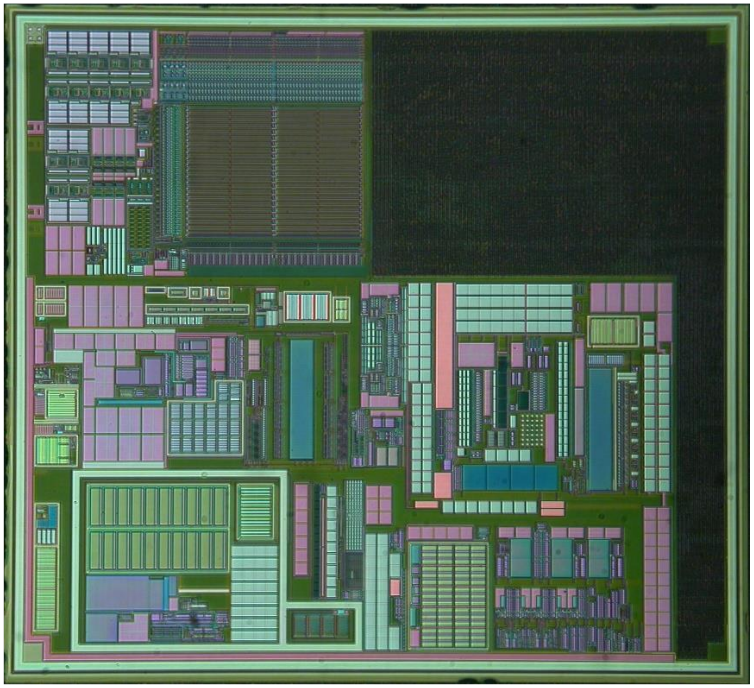
- Requires access to the active area on the chip die with photons
- SLE95250 is fabricated with 90..130nm process and has 5 metal layers
  - there is no anti-tampering sensor mesh on the surface
  - large area is covered with metal and dummy fillers in between
- The only practical way to interfere with the chip operation would be from the rear side of the die using IR laser

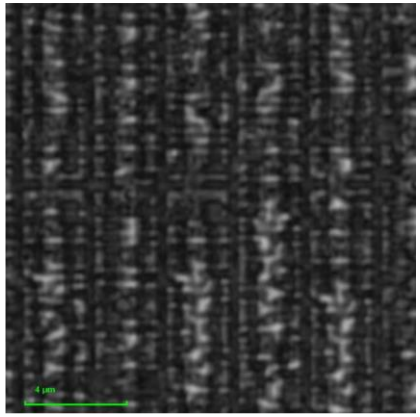


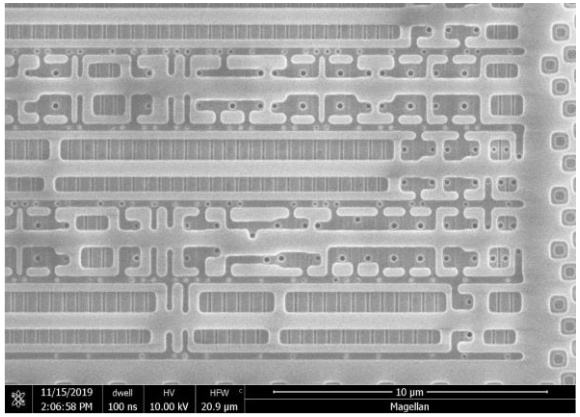


# Optical fault injection

- SLE95250 is fabricated with 90nm process with 5 metal layers
- Photo of fully de-processed die helps with navigation
  - logic area features are beyond the capabilities of optical microscopes (confocal)
  - SEM imaging can be used to create a detailed map of the device, but costly
  - NVM is the best target to inject faults: stores keys and security settings

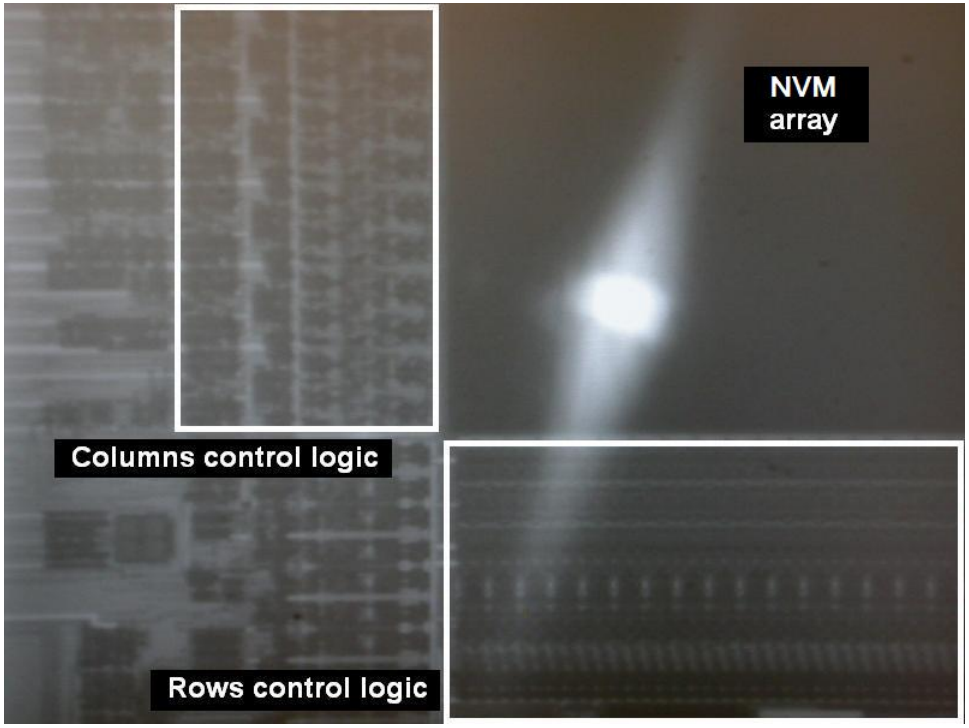






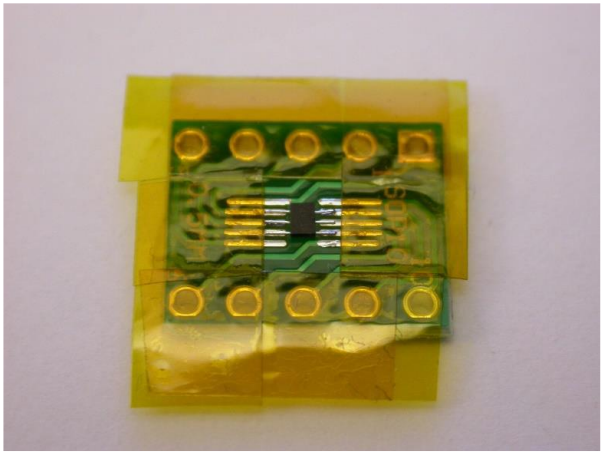
# Injecting faults into NVM

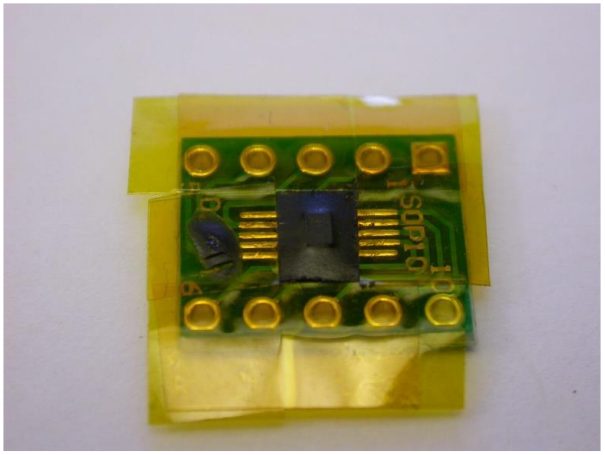
- Locate the area of interest and focus a laser spot at it at the right time
  - aim at a cell: data appear as in erased state
  - aim at a sense amplifier: data appear as in programmed state
  - resolution is limited to ~1μm by the wavelength of the laser (>1000nm)
- Any changes are temporary: as long as the laser is switched on

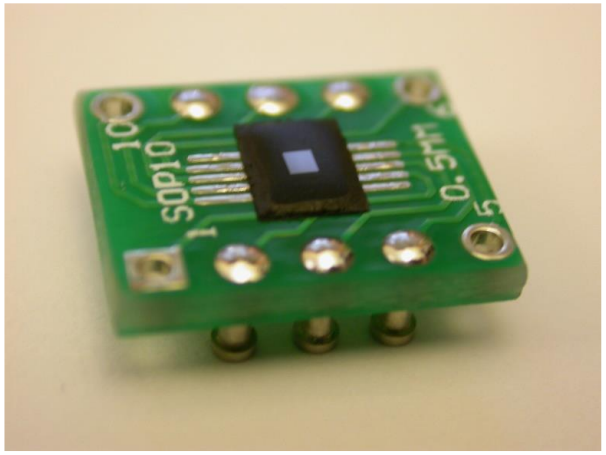


# Injecting faults into NVM

- Only backside approach is effective: simple, inexpensive, no chemicals
- Optiga Trust B is only 1.5mm × 1.1mm
  - soldered to PCB adapter
  - reinforced with epoxy
  - polished on a simple lapping machine
- Fully operational sample with about 20µm left of bulk silicon
  - easy to handle
  - improved optical properties (higher transparency, less beam distortion)







# Injecting faults into NVM

- Only backside approach is effective: simple, inexpensive, no chemicals
- After Hardware Reset the modified security settings are latched

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 A0 86 01 00 00 00 00 00
0050: 04 00 00 00 00 00 6C 1B 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0: 00 00 00 00 00 00 00 00 49 75 A3 7E 70 68 10 0E
00C0: DD 71 D9 B2 03 03 58 D9 CC 3A AC 5C 00 6A A9 F3
00D0: 0C 2F EE FA A6 2F 9C BA 72 68 6E 43 8C EF 77 C7
00E0: 11 CA D0 A4 F1 FA C1 BF 38 02 6D D0 18 BD E1 0D
00F0: F9 13 EA 78 6A AD C9 79 57 3F EC C4 5F A7 20 57
```

```
0000: 00 00 FF FF 00 00 00 00 00 FF 00 00 00 00 00 FF
0010: FF 00 00 00 00 00 FF FF 00 00 00 00 00 FF FF 00
0020: 00 00 00 00 FF FF 00 00 00 FF 00 00 00 00 00 00
0030: 00 00 00 FF 00 00 00 00 00 FF FF 00 00 00 00 FF
0040: FF FF 00 00 00 00 00 FF A0 86 01 00 00 00 00 00
0050: 04 00 00 00 FF FF 6C 1B 00 00 00 FF FF 00 00 00
0060: 00 00 FF FF 00 00 00 00 FF FF FF 00 00 00 00 FF
0070: 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 FF
0080: 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 FF
0090: 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 FF
00A0: 00 00 00 00 00 00 00 FF 00 00 00 00 00 FF FF 00
00B0: 00 00 00 00 00 00 00 00 49 75 A3 7E 70 68 10 0E
00C0: DD 71 D9 FF 03 03 58 D9 CC FF FF 5C 00 6A A9 F3
00D0: FF FF EE FA A6 2F FF FF 72 68 6E 43 8C FF FF C7
00E0: 11 CA D0 A4 F1 FA C1 BF 38 02 FF FF FF BD E1 0D
00F0: F9 FF FF 78 6A AD C9 79 57 FF EC C4 5F A7 20 FF
```

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 5F 79 FE FF FF FF FF FF
0050: 04 00 00 00 00 00 6C 1B 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070: 09 8E 56 98 C4 10 02 3C 20 07 2A 18 77 26 38 5E
0080: 2B 57 CD 1D 90 4C 11 00 C0 9A FF 60 1F F9 C9 57
0090: D9 33 36 2C F8 A5 70 E0 69 3C D3 49 89 8F 80 E4
00A0: EC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0: 00 00 00 00 00 00 00 00 49 75 A3 7E 70 68 10 0E
00C0: DD 71 D9 B2 03 03 58 D9 CC 3A AC 5C 00 6A A9 F3
00D0: 0C 2F EE FA A6 2F 9C BA 72 68 6E 43 8C EF 77 C7
00E0: 11 CA D0 A4 F1 FA C1 BF 38 02 6D D0 18 BD E1 0D
00F0: F9 13 EA 78 6A AD C9 79 57 3F EC C4 5F A7 20 57
```

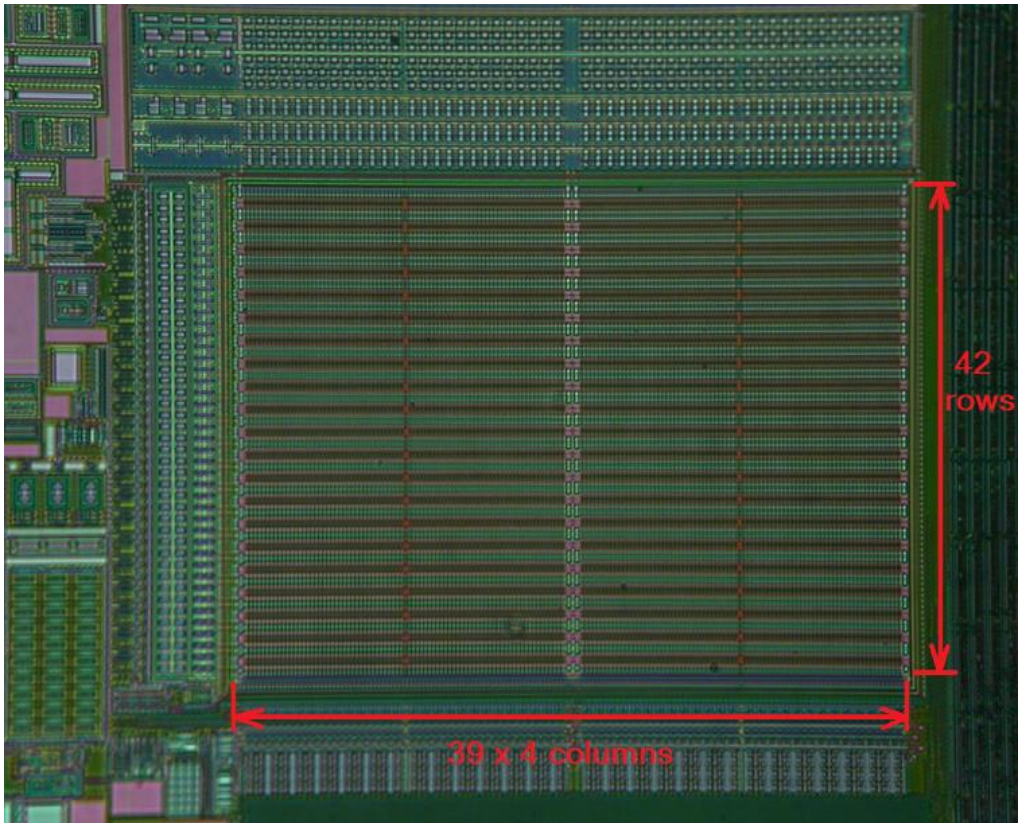
**5F79**… - Inverted Life Span counter area
**098E**… - Device ID
**7726**… - Constant (same in all samples)
**D933**… - ECC curve parameter ($b^{½}$)
**EC** - Unique for each sample

```
[00-3F] user NVM (read and write)
[48-4B] Life Span counter (R/W but lockable)
[50-57] Constants
[B8-E7] ODC: public key Certificate (read only)
[E8-FF] Public Key + nonce (read only)
```

# Reverse engineering of the NVM

- Way of disabling the security is found: gained full access to NVM
- We can read 256 bytes of NVM, but there is no Private Key in that area
- Total size of on-chip NVM is 42×39×4 = 6552 bits

672 bytes of data and 168 bytes of error correction: SECDED Hamming (39 = 32 + 7)



# Quest for backdoors

- Next challenge
  - gain access to all 672 bytes of NVM
  - extract Private Key
  - make 100% clone of the device (same ID, Private/Public key, ODC etc.)
- Sounds like Mission Impossible

  "go there I don't know where and bring it I don't know what"
- Can we reverse engineer the logic without reverse engineering it?
  - we know how to access the registers
  - we know the concept of NVM read/write access
- What else do we need in order to find a backdoor (or Trojan)?
  - Are there any unused bits in existing registers?
  - Are there any additional registers?
  - Are there any registers that behave like known ones?
  - Does security bypassing also unlocks new registers?
  - Any other abnormal behaviour of the device?

# Quest for backdoors

- Scanning the registers space in normal mode
  - R access: [0260…0263] [0268…026E] [026F] [0270] [0272…276] [027D…027F]
  - R/W access: [0260…0263] [026F] [0270] [0272…275] [027D…027F]
- Scanning the registers space in unlocked security
  - R access: [0264] [0266] [0277] [0278]
  - R/W access: [0264] [0266] [0268] [0269] [026B] [026E] [0277] [0278]
- Probing the registers (do a bit of fuzzing)
  - damaged a few dozens of samples, but found interesting registers
  - [0270] NVM mode (charge counter to max, disable device, stop counter)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ? | 0 – count<br>1 – block | ? | ? | ? | 0 – run<br>1 – stop | 0 – stp wr<br>1 – wrt 0s | ? |

  - [0275] NVM write protection (user NVM area)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 – norm<br>1 – 38-3F | 0 – norm<br>1 – 30-37 | 0 – norm<br>1 – 28-2F | 0 – norm<br>1 – 20-27 | 0 – norm<br>1 – 18-1F | 0 – norm<br>1 – 10-17 | 0 – norm<br>1 – 08-0F | 0 – norm<br>1 – 00-07 |

# Quest for backdoors

- Probing the registers (further damage of samples)

  [026F] NVM security (counter write protection, read protection, full write protection)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ? | 0 – norm<br>1 – WP all | ? | ? | 0 – no RP<br>1 – RP | 0 – norm<br>1 – WP C | ? | ? |

- Additional functions in unlocked security (no RP), extended NVM

  [0264] ENVM control (data encryption, erase row)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 – norm<br>1 – erase | ? | ? | ? | ? | ? | ? | 0 – encr<br>1 – array |

  [0266] ENVM command (set address, read/write, start [WR]/status[RD])

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 – ready<br>1 – start | 0 – read<br>1 – write | NVM address [9:4] | | | | | |

# Quest for backdoors

- Additional functions in unlocked security (no RP): new functions

  [0270] NVM mode (charge counter to max, disable device, direct write of EC code)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ? | 0 – norm<br>1 – EC wr | ? | ? | ? | 0 – run<br>1 – stop | 0 – stp wr<br>1 – wrt 0s | ? |

- Data Buffers

  [0010 – 001F] ENVM read/write buffer

  [0020 – 0023] Error Correction Code read/write buffer

- Extended NVM read (all 672 bytes of data and 168 bytes of EC code)

  820, 851, 502, 666, 4xx XX is 0x80+Addr[9:4]

  820, 851, 5xx, 7yy, 7zz/7zz YY:XX address of NVM read buffer, ZZ is data

- Extended NVM write

  820, 851, 5xx, 6yy, 4zz/4zz YY:XX address of NVM write buffer, ZZ is data

  820, 851, 502, 666, 4xx XX is 0xC0+Addr[9:4]

  820, 851, 502, 666, 7xx XX bit 7 is status (0 – ready)

# Memory map of the Extended NVM

```
0000: 00 00 00 00 01 01 01 01 00 00 00 00 00 00 00 00  00 24 00 00
0010: 02 02 02 02 03 03 03 03 00 00 00 00 00 00 00 00  3F 1B 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0090: 5F 79 FE FF FF FF FF FF 00 00 00 00 00 00 00 00  6C 60 00 00
00A0: 04 00 00 00 00 00 90 13 00 00 00 00 00 00 00 00  19 0B 00 00
00B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
00E0: 68 0D 22 98 C4 10 02 3C 00 00 00 00 00 00 00 00  7C 13 00 00
00F0: 20 07 2A 18 77 26 38 5E 00 00 00 00 00 00 00 00  68 7A 00 00
0100: 2B 57 CD 1D 90 4C 11 00 00 00 00 00 00 00 00 00  22 67 00 00
0110: C0 9A FF 60 1F F9 C9 57 00 00 00 00 00 00 00 00  2D 18 00 00
0120: D9 33 36 2C F8 A5 70 E0 00 00 00 00 00 00 00 00  4B 68 00 00
0130: 69 3C D3 49 89 8F 80 E4 00 00 00 00 00 00 00 00  5F 13 00 00
0140: A3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  12 00 00 00
0150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0170: 36 6A 93 2E FE 0D B8 16 00 00 00 00 00 00 00 00  09 04 00 00
0180: EC A4 CA 4A 62 D5 8A 77 00 00 00 00 00 00 00 00  34 0B 00 00
0190: C3 53 21 09 50 FA 41 82 00 00 00 00 00 00 00 00  56 77 00 00
01A0: 6F 54 13 CD DC 24 32 C1 00 00 00 00 00 00 00 00  4C 4C 00 00
01B0: D6 F9 02 D8 CA 51 C5 DA 00 00 00 00 00 00 00 00  60 7E 00 00
01C0: C7 3D EF D5 C0 77 5A BB 00 00 00 00 00 00 00 00  12 7B 00 00
01D0: B4 38 E6 E4 12 DB 3B 29 00 00 00 00 00 00 00 00  65 69 00 00
01E0: 79 4F 67 A8 C4 AF F7 92 00 00 00 00 00 00 00 00  06 0B 00 00
01F0: 1D A5 EB EE FA A6 B9 05 00 00 00 00 00 00 00 00  1E 1C 00 00
0200: 8E A5 8E A5 95 D6 95 D6 0A 5E 91 58 48 9C 13 E6  65 69 6E 3A
0210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0220: 8E A5 8E A5 95 D6 95 D6 0C 56 E6 6F 8E A5 8E A5  65 69 40 65
0230: 31 BC 31 BC 5C 96 5C 96 1A 20 09 63 32 25 2C 31  3C 11 24 09
0240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0250: 8E A5 8E A5 95 D6 95 D6 B3 9C 4D 83 10 2C 95 A8  65 69 55 04
0260: 31 BC 31 BC 5C 96 5C 96 EC BF DC E2 2D 6A 27 13  3C 11 0D 02
0270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0280: 00 00 00 00 00 00 00 00 31 1C 31 BC 5C 96 5C C6  00 00 54 60
0290: 8E A5 8E A5 86 AC 8E A5 95 D6 95 D6 95 D6 95 D6  65 04 69 69
```

**5F79… - Inverted Life Span counter**
**680D… - Device ID**
**7726… - Constants (same in all samples)**
**D933… - ECC curve parameter ($b^{1/2}$)**
**A300… - Unique number for each device**
**366A… - ODC: public key Certificate**
**B438… - Public Key + nonce**
**8EA5… - Encrypted data**

Only the first 8 bytes are used in [000-1FF]
[000-077] user NVM
[090-093] Life Span counter
[0A0-0A7] Constants
[0E0-0F3] Device ID
[0F4-117] Constants
[120-137] ECC curve parameter
[140-140] Memory Encryption key
[170-1C7] ODC: public key Certificate
[1D0-1F7] Public Key + nonce
[200-29F] Encrypted data

# Further quest for backdoors

- Hamming code in ENVM
  - polinomial coefficients can be found by programming 00..01, 00..02, 00..04,…, 80..00
  - Error Correction Code can be overwritten (register [0270] bit 6 controls this)
  - single errors are correctable, double errors result in FF value read in NVM mode
- Memory encryption and decryption
  - unique for each device and affected by NVM value at [A0] (ENVM at [140])
  - register [0264] bit 0 enables decryption of area 0200-029F
  - register [0278] contains decryption key, but it is only 8-bit long
  - it can be brute forced within seconds
- Decryption key
  - register [0277] contains the copy of device's unique number
  - on Reset the decryption key is derived from the unique number and stored in register
  - there is no need to brute force it – just configure the ENVM control registers correctly

# Memory map of decrypted ENVM

- Private key extraction and verification
  - Read ENVM with correct settings in registers [0264] and [0278]
  - compute $q \cdot G$ and compare with $Q$ ($G$ – base point, $q$ – private key, $Q$ – public key)
  - ECC computation ends with timeout if the private key is modified
  - CRC of the Private key is stored in ENVM
  - CRC is a linear function: $CRC_1$ *xor* $CRC_2 = CRC_3$, $Key_1$ *xor* $Key_2 = Key_3$

```
0200: 8E A5 8E A5 95 D6 95 D6 0A 5E 91 58 48 9C 13 E6  65 69 6E 3A
0210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0220: 8E A5 8E A5 95 D6 95 D6 0C 56 E6 6F 8E A5 8E A5  65 69 40 65
0230: 31 BC 31 BC 5C 96 5C 96 1A 20 09 63 32 25 2C 31  3C 11 24 09
0240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0250: 8E A5 8E A5 95 D6 95 D6 B3 9C 4D 83 10 2C 95 A8  65 69 55 04
0260: 31 BC 31 BC 5C 96 5C 96 EC BF DC E2 2D 6A 27 13  3C 11 0D 02
0270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
0280: 00 00 00 00 00 00 00 00 31 1C 31 BC 5C 96 5C C6  00 00 54 60
0290: 8E A5 8E A5 86 AC 8E A5 95 D6 95 D6 95 D6 95 D6  65 04 69 69
```

8EA5… - Encrypted data

```
0200: 00 00 00 00 89 09 B7 93 00 00 00 00 00 00 00 00  65 40 65 69
0210: B9 2C 83 FD E3 6B 7A 07 00 00 00 00 00 00 00 00  24 09 3C 11
0220: CD 16 61 FA 09 B2 47 08 00 00 00 00 00 00 00 00  02 0D 3C 11
0230: D9 33 36 2C F8 A5 70 E0 00 00 00 00 00 00 00 00  55 04 65 69
0240: 69 3C D3 49 89 8F 80 E4 00 00 00 00 00 00 00 00  3A 6E 65 69
0250: 1B 3C 13 CB C6 59 69 C5 1B 3C 13 CB C6 59 69 C5  00 00 00 00
0260: 1B 3C 13 CB C6 59 69 C5 1B 3C 13 CB C6 59 69 C5  00 00 00 00
0270: 89 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04 69 65 69
0280: 00 00 05 00 0A 00 00 00 E5 A8 E5 A8 D9 65 D9 65  60 54 00 00
0290: 1B 3C 13 CB C6 59 69 C5 1B 3C 13 CB C6 59 69 C5  00 00 00 00
```

8909… - CRC of Encryption key
B92C… - Private Key
D933… - ECC curve parameter ($b^{½}$)
8900… - Decryption Key
0000… - Security settings

```
[204-207] CRC of the private key
[210-227] Private Key
[230-247] ECC curve parameter
[250-26F] Decrypted 00..00
[270-270] Decryption Key
[280-287] Security settings
[200-29F] Decrypted 00..00
```

# Secrets from one compromised device

- Public key

  Qx = **0x06d046e3bf7bb34479bd3aad1301f14cbd**

  Qx* = **0x1dd6d046e3bf7bb34479bd3aad1301f14cbd**

- Device ID

  D = **0x07203c0210c4981a8d68**

- Signature

  r = **0x001c8f15507787ba50c293427d0794f447e899c150**

  s = **0x00167334723255207c535908434ac0563548dbaa1d**

- Recovered Secret key (128-bit)

  q = **0xd861429f79fefd9f8090ae83df804970**

- Real Secret key (131-bit with 3 most significant bits equal 0)

  q = **0x0d861429f79fefd9f8090ae83df804970**

# Quest for backdoors

- Memory is also scrambled (address lines permutation)
  - part of the memory encryption function
  - EC codes are not encrypted and follow the scrambled data
- Memory encryption is XOR function
  - enc(0) XOR enc(N) = N
- Making 1:1 clone

1) disable security protection
2) read all the plaintext data from NVM at [00…FF]
3) make correct settings in ENVM registers
4) read decrypted ENVM data
5) get blank device with the same part number (SLE95250)
6) disable security protection
7) write NVM data
8) reset the device to activate encryption
9) write plaintext ENVM data (automatically encrypted)
10) enjoy another Optiga Trust B device with the same ID, Keys, data and security

# Limitations and improvements

- The attack time is substantial and requires qualified person to perform
  - dedicated PCB adapters
  - device needs to be soldered to the adapter
  - encapsulation needed around the edges
  - precision polishing/lapping to remove package and polish the silicon die
  - dedicated optical fault injection setup with IR laser
  - need to design and fabricate of substitution devices
- Side-channel attacks could be faster
  - improve synchronisation and reduce noise
  - find more efficient way for an attack: DPA, CPA, Template etc.
  - still the need to design and fabricate of substitution devices
- Can we find a major security flaw that would allow ultimate access?
  - reduce the cost and time of an attack by 100…1000 times
  - reduce the cost of re-implementation by a factor of 10 (no need for substitution)

# ~~Future~~ Past Work

- Hardware Security in semiconductor devices
  - low-level security critical features are implemented in silicon
  - security critical features are controlled by logic gates hardwired in silicon
  - many features are supplied as black boxes with known input and output
  - firmware does not have much control over the hardware process flow
- Formal security evaluation is unlikely to spot process variations
  - fabrication of many semiconductor devices is outsourced
  - low level structures, parameters and layout might change
  - developer has limited knowledge about some structures that could impact the security
- It was possible to find a way to disable the security in SLE95250
  - purely Non-Invasive attack: no need to even de-solder the device from board
  - very fast process and fully reversible: no evidence of the attack
  - time: 0.1s to disable security, 0.2s to read all data, 0.6s to copy into a new device
  - Infineon was notified about the security flaw
  - full paper and a conference talk to appear by the end of 2020

# Conclusion

- Optiga Trust B is reverse engineered without any NDA
  - full authentication process is completely replicated
  - all information from embedded NVM is extracted (672 bytes + 168 bytes EC code)
  - fully working clone is created with same ID, private&public key, ODC, encrypted etc.
- Hardware Security has demonstrated its importance
  - the gap between hardware and software is widening
  - no direct control over security-critical components
- Hardware Security cannot rely on obscurity and lack of information
- Many semiconductor devices have backdoors (or Trojans?)
- Determined attacker could overcome any protection: cost and time
- Hardware Security can help with counterfeit prevention if used properly
- New approaches and methods are essential in fighting modern challenges and are likely to be developed
- Stay tuned to learn how to clone Optiga Trust B in less than 1 second

# Thank you!

URL: *http://www.cst.cam.ac.uk/~sps32*

email: *sps32@cam.ac.uk*