Compromising devices security via NVM controller vulnerability

Dr Sergei Skorobogatov

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

UNIVERSITY OF CAMBRIDGE
Dept of Computer Science and Technology

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
- Some of the research achievements with significant impact

2002: discovery of optical fault injection attacks shook the semiconductor industry

2005: prove of data remanence in EEPROM and Flash memory

2006: introduction of powerful combined attacks of fault injection with power analysis

2010: bumping attacks that can extract AES key and data from protected Flash memory

2012: hardware acceleration of power analysis for finding backdoors

2016: demonstration of “impossible” NAND mirroring attack on iPhone 5c

2016: direct SEM imaging of EEPROM and Flash memory contents

2018: live decapsulation carried on a battery powered chip

Authentication devices: 1980s…today

- Security via obscurity – 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:** side-channel, fault injection, reverse engineering
- 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
  - each device stores unique **key shared with host devices**
  - 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 not shared with anyone**
  - 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

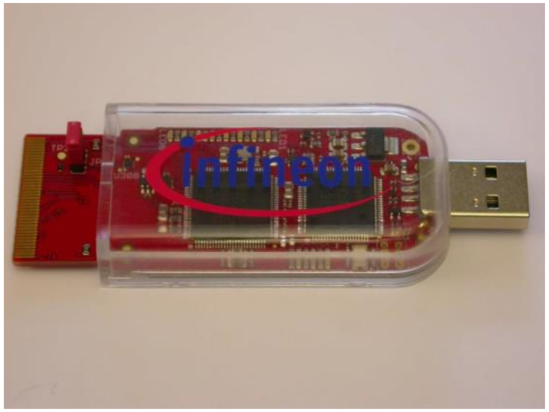
- Texas Instruments: BQ40Z80
  - devices with documentation and evaluation/development kits are available
- Maxim Semiconductors: DS28C36, DS28E36, DS28E38
  - devices and evaluation kits with documentation are available
  - datasheets and libraries can be found
- Microchip(former Atmel): ATECC508A, 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 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

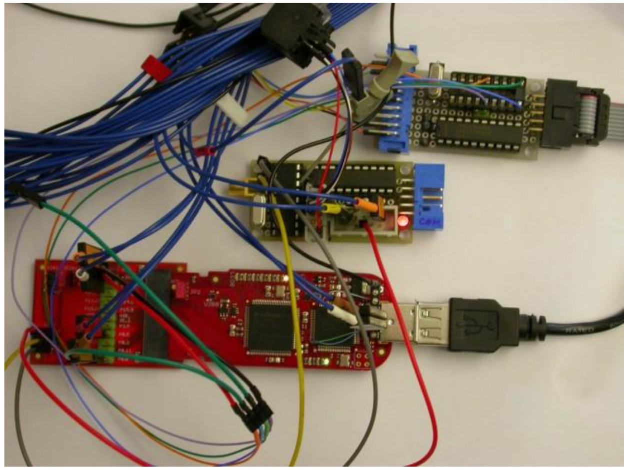
Optiga™ Trust B Evaluation Kit

- 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

- 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



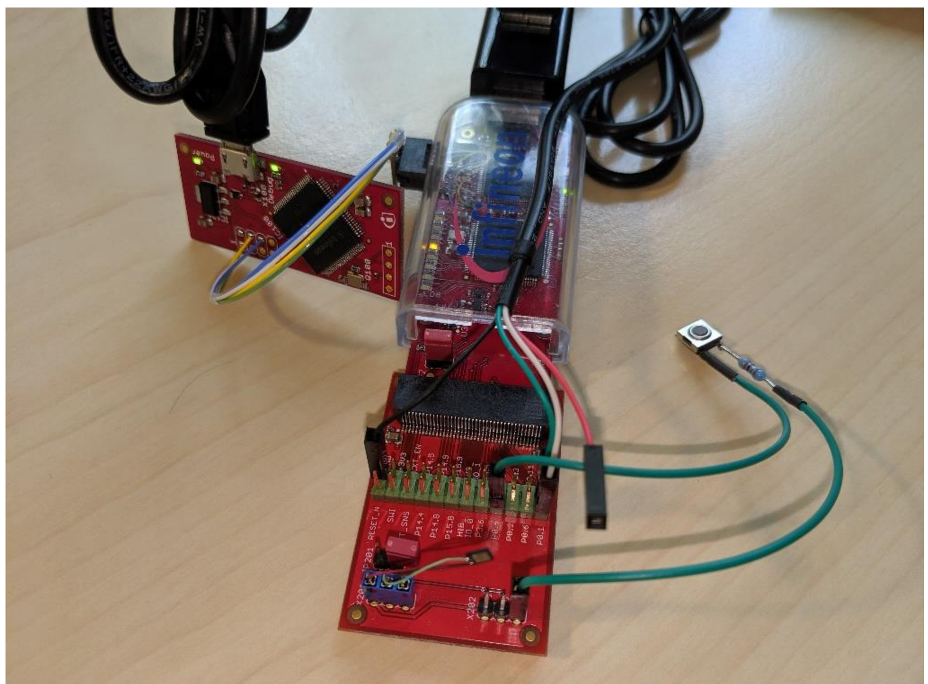
```
********************************************************************************
Infineon Technologies Origa1&2 Console Test Programs
Code Compiled: Mar  3 2017 15:04:50
Code Version: 3.2.0
RTC: 0:0:0
Note: Using TeraTerm, change the setting at Setup->Terminal->New-line.>Transmit:CR+LF
*********************************************************************************
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.
```

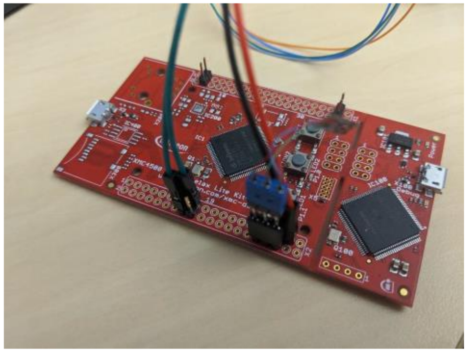
Reverse engineering of the Evaluation Kit

- 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 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 1 – start	0 – read 1 – write	select buffer	NVM address [7:3]				

[0272] NVM command

7	6	5	4	3	2	1	0
?	?	0 – direct 1 – count	length, bytes: 00 – 1, 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

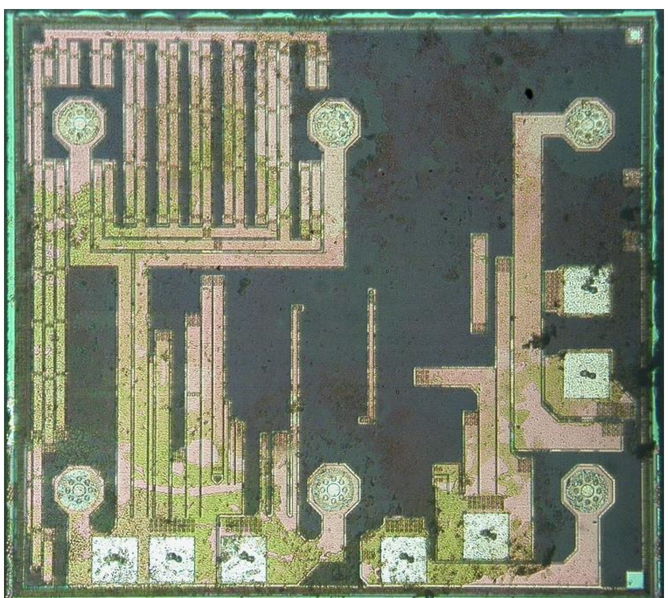
820, 851, 5xx, 6yy, 4zz/4zz	YY:XX address of NVM write buffer, ZZ is data
820, 851, 502, 674, 4xx	XX is Addr[2:0]
820, 851, 502, 672, 4xx	XX is 0xC0+Addr[7:3]
820, 851, 502, 672, 7xx	XX bit 7 is status (0 – ready)

- 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)

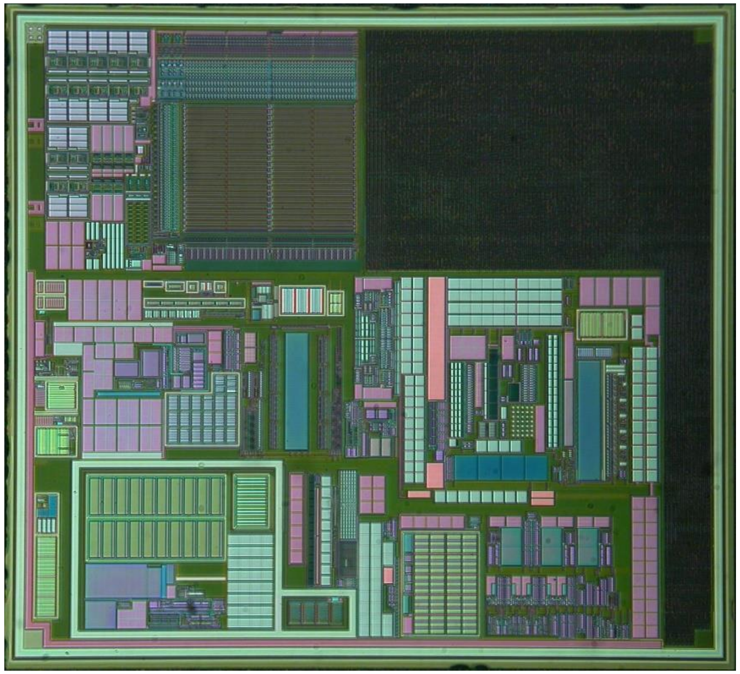
Optical fault injection

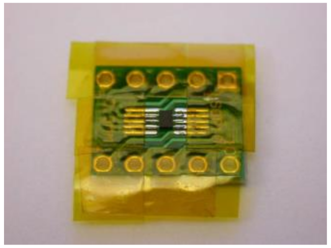
- Requires access to the active area on the chip die with photons
- SLE95250 is fabricated with 90..130nm process and has 5 metal layers
  - there is no anti-tampering sensor mesh on the surface
  - large area is covered with metal and dummy fillers in between
- The only practical way to interfere with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

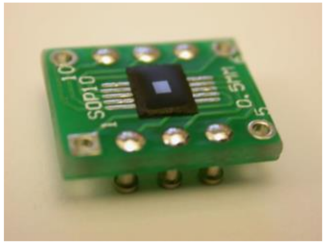


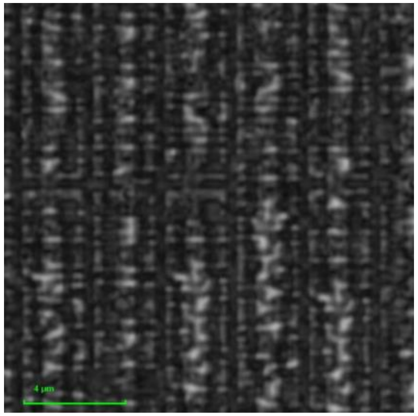
Optical fault injection

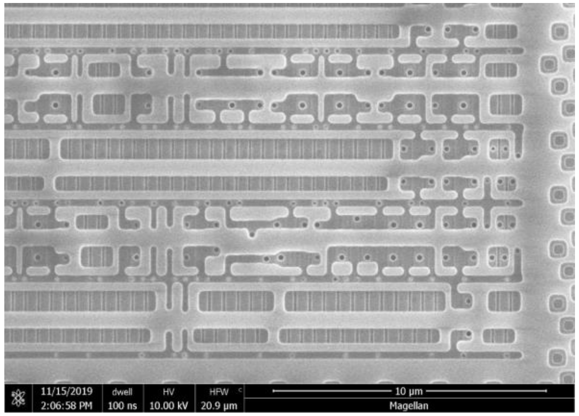
- Backside approach is the only practical way
  - photo of fully de-processed die helps with navigation
  - challenging sample preparation requires package reinforcing
  - logic area features are beyond the capabilities of optical microscopes (confocal)
  - SEM imaging can be used to create a detailed map of the device, but costly
  - NVM is the best target to inject faults: stores keys and security settings





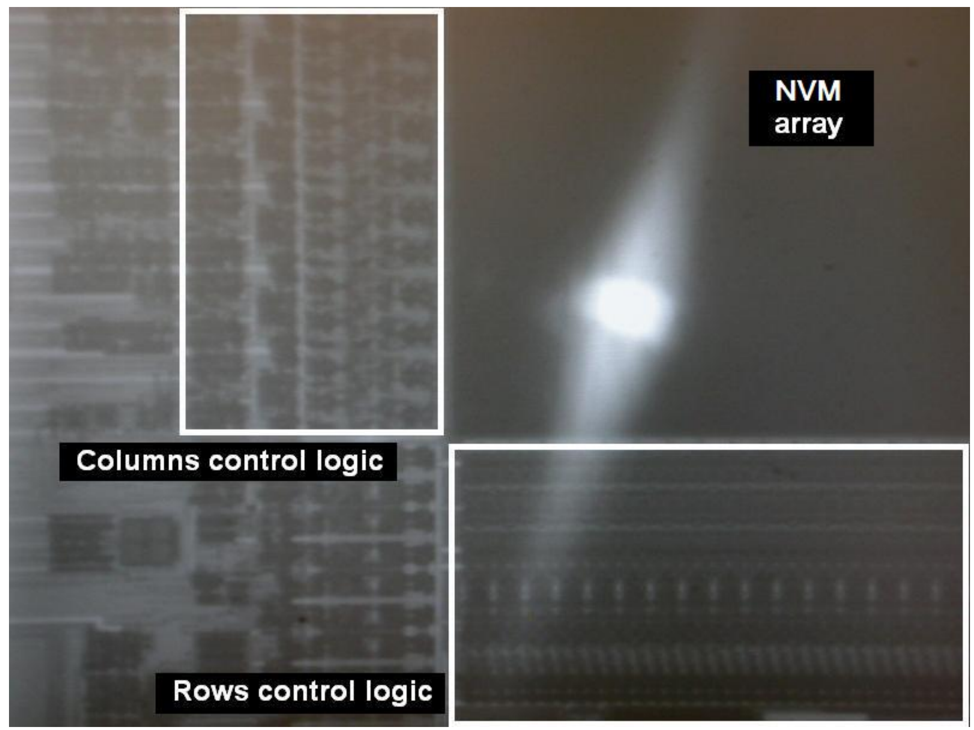






Injecting faults into NVM

- 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
- 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 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
```

```
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
```

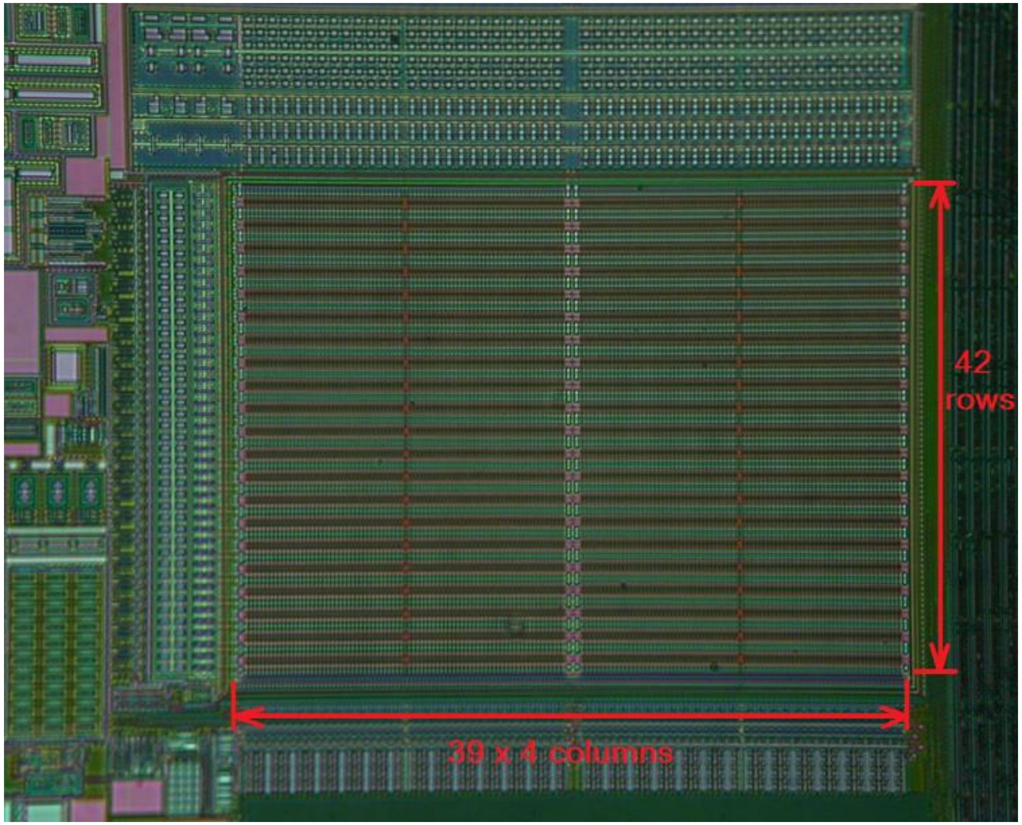
5F79… – Inverted Life Span counter area
098E… – Device ID
7726… – Constant (same in all samples)
D933… – ECC curve parameter ($b^{1/2}$)
EC – Unique for each sample

[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 $42 \times 39 \times 4 = 6552$ 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 1 – block	?	?	?	0 – run 1 – stop	0 – stp wr 1 – wrt 0s	?

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

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0 – norm 1 – 38-3F	0 – norm 1 – 30-37	0 – norm 1 – 28-2F	0 – norm 1 – 20-27	0 – norm 1 – 18-1F	0 – norm 1 – 10-17	0 – norm 1 – 08-0F	0 – norm 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 1 – WP all	?	?	0 – no RP 1 – RP	0 – norm 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 1 – erase	?	?	?	?	?	?	0 – encr 1 – array

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

7	6	5	4	3	2	1	0
0 – ready 1 – start	0 – read 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 1 – EC wr	?	?	?	0 – run 1 – stop	0 – stp wr 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 ($b^{½}$)
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 encryption is XOR function: enc(0) XOR enc(N) = N
  - EC codes are not encrypted and follow the scrambled data

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
```

```
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
```

8EA5… - Encrypted data

8909… - CRC of Encryption key

B92C… - Private Key

D933… - ECC curve parameter ($b^{1/2}$)

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**

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)

NVM operation and security

- NVM can be programmed by bits but erased by rows
- Conventional NVM memory (EEPROM or Flash) has inherent security
  - writing can change single bit, but only in one direction (‘1’ → ‘0’)
  - erasing is a totally different operation at hardware level (multiple bits ‘0’ → ‘1’)
  - OTP mode (no erasing) permits the security to be changed only from low to high
- NVM in modern chips with advanced fabrication process (28nm…90nm)
  - small cell size (high density, large arrays)
  - fast programming and erasing (high throughput)
  - maximum number of programming cycles (limit number of overwritings)
  - reduced data retention time (shorter storage time)
  - reduced yield in production (dead cells)
- Improving NVM parameters
  - testing and optimising physical array
  - correcting errors
  - store multiple copies of data

Exploiting NVM vulnerability

- Hardware Security in semiconductor devices with embedded NVM
  - 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
- Writing to NVM
  - data from specific row in the memory array is stored in a buffer
  - buffer content is modified
  - array row erase operation is started and internally timed
  - row writing from the buffer is performed and internally timed
  - memory busy bit in status register is changed to ‘not busy’
  - mind the Smart Buffer: no overwriting for the same data

duration µs	**0**	**58**	**77**	**78**	**79**	**830**	**831**	**832**	**833**	**999**
Value	**5A**	**5A**	**7B**	**FB**	**FF**	**FF**	**F7**	**A7**	**A5**	**A5**

Exploiting NVM vulnerability

- Hardware approach (power glitching)
  - change the security level (lock CNT) or impose write protection on some user data
  - wait for pre-determined time t_1 to allow the erasure of specific security bits
  - power down the device by shorting V_{CC} to GND
  - recover the device security by changing write protection level (restore row ECC)
- Software approach (self-induced fault)
  - change the security level (lock CNT) or impose write protection on some user data
  - wait for pre-determined time t_2 to allow the erasure of specific security bits
  - set bit 2 in 0x270 register to activate a kill switch
  - recover the device security by changing write protection level (restore row ECC)
- Results
  - successful Non-Invasive attack on Optiga™ Trust B in less than 0.1 seconds
  - no need to de-solder the chip thanks to soft-kill-switch
  - fully reversible: no evidence of the attack
  - complete device cloning in less than 1 second

Countermeasures

- Separate NVM arrays for system, user and security
  - significant penalty for area: in small arrays 90% will be used by control logic
  - could give some clues to the attacker about the security location and its logic
- CRC checks
  - prevent data manipulation with relatively low overheads
  - can be bypassed if the attacker can overwrite the memory locations
- Redundancy
  - more robust error correction
  - store multiple copies of the configuration and security data
- Combined approach
  - proper memory partitioning
  - data encryption
  - CRC check
  - multiple copies of data

Conclusion

- Optiga^{TM} 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.
  - very fast (<1s) Non-Invasive attack found: no need de-solder the device from board
  - Infineon was notified about the security flaw in SLE95250
- Hardware Security has demonstrated its importance
  - the gap between hardware and software is widening
  - no direct control over security-critical components
  - formal security evaluation is unlikely to spot process variations
- 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
- New approaches and methods are essential in fighting modern challenges and are likely to be developed

Thank you!

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

email: *sps32@cam.ac.uk*