Compromising devices security via NVM controller vulnerability

Dr Sergei Skorobogatov

http://www.cst.cam.ac.uk/~sps32 email: sps32@cam.ac.uk
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
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
## 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])
  
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – ready</td>
<td>0 – read</td>
<td>select buffer</td>
<td>NVM address [7:3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 – start</td>
<td>1 – write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- [0272] NVM command
  
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>0 – direct</td>
<td>length, bytes: 00 – 1, 01 – 2, 10 – 4, 11 – 8</td>
<td>NVM address [2:0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – count</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 ~1\(\mu\)m 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 00 00 00 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

5F79... - Inverted Life Span counter area
098E... - Device ID
7726... - Constant (same in all samples)
D933... - ECC curve parameter (b^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)

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>?</td>
<td>0 – count</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0 – run</td>
<td>0 – stp wr</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – block</td>
<td></td>
<td></td>
<td></td>
<td>1 – stop</td>
<td>1 – wrt 0s</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 – norm</td>
<td>0 – norm</td>
<td>0 – norm</td>
<td>0 – norm</td>
<td>0 – norm</td>
<td>0 – norm</td>
<td>0 – norm</td>
<td>0 – norm</td>
</tr>
<tr>
<td></td>
<td>1 – 38-3F</td>
<td>1 – 30-37</td>
<td>1 – 28-2F</td>
<td>1 – 20-27</td>
<td>1 – 18-1F</td>
<td>1 – 10-17</td>
<td>1 – 08-0F</td>
<td>1 – 00-07</td>
</tr>
</tbody>
</table>
**Quest for backdoors**

- **Probing the registers (further damage of samples)**
  
  [026F]  
  NVM security (counter write protection, read protection, full write protection)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>0 – norm</td>
<td>?</td>
<td>?</td>
<td>0 – no RP</td>
<td>0 – norm</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>1 – WP all</td>
<td></td>
<td></td>
<td>1 – RP</td>
<td>1 – WP C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – erase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 – array</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – ready</td>
<td>0 – read</td>
<td>NVM address [9:4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 – start</td>
<td>1 – write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>0 – norm</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0 – run</td>
<td>0 – stp wr</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>1 – EC wr</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>1 – stop</td>
<td>1 – wrt 0s</td>
<td>?</td>
</tr>
</tbody>
</table>
```

- 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

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Public Key + nonce</td>
</tr>
<tr>
<td>0010</td>
<td>Memory Encryption key</td>
</tr>
<tr>
<td>0020</td>
<td>Constants</td>
</tr>
<tr>
<td>0030</td>
<td>ODC: public key Certificate</td>
</tr>
<tr>
<td>0040</td>
<td>ECC curve parameter (b&lt;sup&gt;59&lt;/sup&gt;)</td>
</tr>
<tr>
<td>0050</td>
<td>Unique number for each device</td>
</tr>
<tr>
<td>0060</td>
<td>-</td>
</tr>
<tr>
<td>0070</td>
<td>-</td>
</tr>
<tr>
<td>0080</td>
<td>-</td>
</tr>
</tbody>
</table>
| 0090   | 5F 79 FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
Further quest for backdoors

• Hamming code in ENVM
  – polynomial 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: $CRC_1 \ xor \ CRC_2 = CRC_3$, $Key_1 \ xor \ Key_2 = Key_3$

<table>
<thead>
<tr>
<th>Offset</th>
<th>Data</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0200:</td>
<td>8E A5 8E A5 95 D6 95 D6 0A 5E 91 58 48 9C 13 E6</td>
<td>8EA5... - Encrypted data</td>
</tr>
<tr>
<td>0210:</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>8909... - CRC of Encryption key</td>
</tr>
<tr>
<td>0220:</td>
<td>08 A5 8E A5 95 D6 95 D6 0C 56 E6 5F 8E A5 8E A5</td>
<td>B92C... - Private Key</td>
</tr>
<tr>
<td>0230:</td>
<td>31 BC 31 BC 5C 96 5C 96 1A 20 09 63 32 25 2C 31</td>
<td>D933... - ECC curve parameter ($b^\frac{1}{2}$)</td>
</tr>
<tr>
<td>0240:</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>8900... - Decryption Key</td>
</tr>
<tr>
<td>0250:</td>
<td>08 A5 8E A5 86 AC 8E A5 95 D6 95 D6 95 D6 95 D6</td>
<td>0000... - Security settings</td>
</tr>
<tr>
<td>0260:</td>
<td>8E A5 8E A5 89 09 B7 93 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>0270:</td>
<td>0B 9C 83 FD E3 6B 7A 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>[204-207] CRC of the private key</td>
</tr>
<tr>
<td>0280:</td>
<td>0D 16 61 FA 09 B2 47 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>[210-227] Private Key</td>
</tr>
<tr>
<td>0290:</td>
<td>08 A5 8E A5 86 AC 8E A5 95 D6 95 D6 95 D6 95 D6</td>
<td>[230-247] ECC curve parameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Data</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0200:</td>
<td>8E A5 8E A5 95 D6 95 D6 0A 5E 91 58 48 9C 13 E6</td>
<td>8EA5... - Encrypted data</td>
</tr>
<tr>
<td>0210:</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>8909... - CRC of Encryption key</td>
</tr>
<tr>
<td>0220:</td>
<td>08 A5 8E A5 95 D6 95 D6 0C 56 E6 5F 8E A5 8E A5</td>
<td>B92C... - Private Key</td>
</tr>
<tr>
<td>0230:</td>
<td>31 BC 31 BC 5C 96 5C 96 1A 20 09 63 32 25 2C 31</td>
<td>D933... - ECC curve parameter ($b^\frac{1}{2}$)</td>
</tr>
<tr>
<td>0240:</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>8900... - Decryption Key</td>
</tr>
<tr>
<td>0250:</td>
<td>08 A5 8E A5 86 AC 8E A5 95 D6 95 D6 95 D6 95 D6</td>
<td>0000... - Security settings</td>
</tr>
<tr>
<td>0260:</td>
<td>8E A5 8E A5 89 09 B7 93 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>0270:</td>
<td>0B 9C 83 FD E3 6B 7A 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>[204-207] CRC of the private key</td>
</tr>
<tr>
<td>0280:</td>
<td>0D 16 61 FA 09 B2 47 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>[210-227] Private Key</td>
</tr>
<tr>
<td>0290:</td>
<td>08 A5 8E A5 86 AC 8E A5 95 D6 95 D6 95 D6 95 D6</td>
<td>[230-247] ECC curve parameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Data</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0200:</td>
<td>8E A5 8E A5 95 D6 95 D6 0A 5E 91 58 48 9C 13 E6</td>
<td>8EA5... - Encrypted data</td>
</tr>
<tr>
<td>0210:</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>8909... - CRC of Encryption key</td>
</tr>
<tr>
<td>0220:</td>
<td>08 A5 8E A5 95 D6 95 D6 0C 56 E6 5F 8E A5 8E A5</td>
<td>B92C... - Private Key</td>
</tr>
<tr>
<td>0230:</td>
<td>31 BC 31 BC 5C 96 5C 96 1A 20 09 63 32 25 2C 31</td>
<td>D933... - ECC curve parameter ($b^\frac{1}{2}$)</td>
</tr>
<tr>
<td>0240:</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>8900... - Decryption Key</td>
</tr>
<tr>
<td>0250:</td>
<td>08 A5 8E A5 86 AC 8E A5 95 D6 95 D6 95 D6 95 D6</td>
<td>0000... - Security settings</td>
</tr>
<tr>
<td>0260:</td>
<td>8E A5 8E A5 89 09 B7 93 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
</tr>
</tbody>
</table>
Secrets from one compromised device

- Public key
  \[ Q_x = 0x06d046e3bf7bb34479bd3aad1301f14cbd \]
  \[ Q_x^* = 0x1dd6d046e3bf7bb34479bd3aad1301f14cbd \]

- 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

<table>
<thead>
<tr>
<th>duration (µs)</th>
<th>0</th>
<th>58</th>
<th>77</th>
<th>78</th>
<th>79</th>
<th>830</th>
<th>831</th>
<th>832</th>
<th>833</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5A</td>
<td>5A</td>
<td>7B</td>
<td>FB</td>
<td>FF</td>
<td>FF</td>
<td>F7</td>
<td>A7</td>
<td>A5</td>
<td>A5</td>
</tr>
</tbody>
</table>
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™ 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