183.222
Advanced Internet Security

Hardware Security

Adrian Dabrowski
Georg Merzdovnik
Aljosha Judmayer
Christian Kudera

News from the Field: VirtualBox 0day

- On Nov 6, Sergej Zeleniuk published (full disclosure - 0day) details of a critical guest-to-host escape vulnerability
  - Vulnerable software: VirtualBox 5.2.20 and prior versions
  - Host OS: any, the bug is in a shared code base
  - Guest OS: any
  - VM configuration: default (the only requirement is that a network card is Intel PRO/1000 MT Desktop (82540EM) and a mode is NAT)
  - Details: https://github.com/MorteNoir1/virtualbox_e1000_0day

News from the Lab

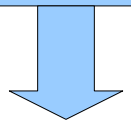
- Deadline for the GNURadio Challenge
  - 22.11.2018, 00:00

- So far only 4 students solved the challenge ...

Introduction to Embedded System Security

A typical Embedded System

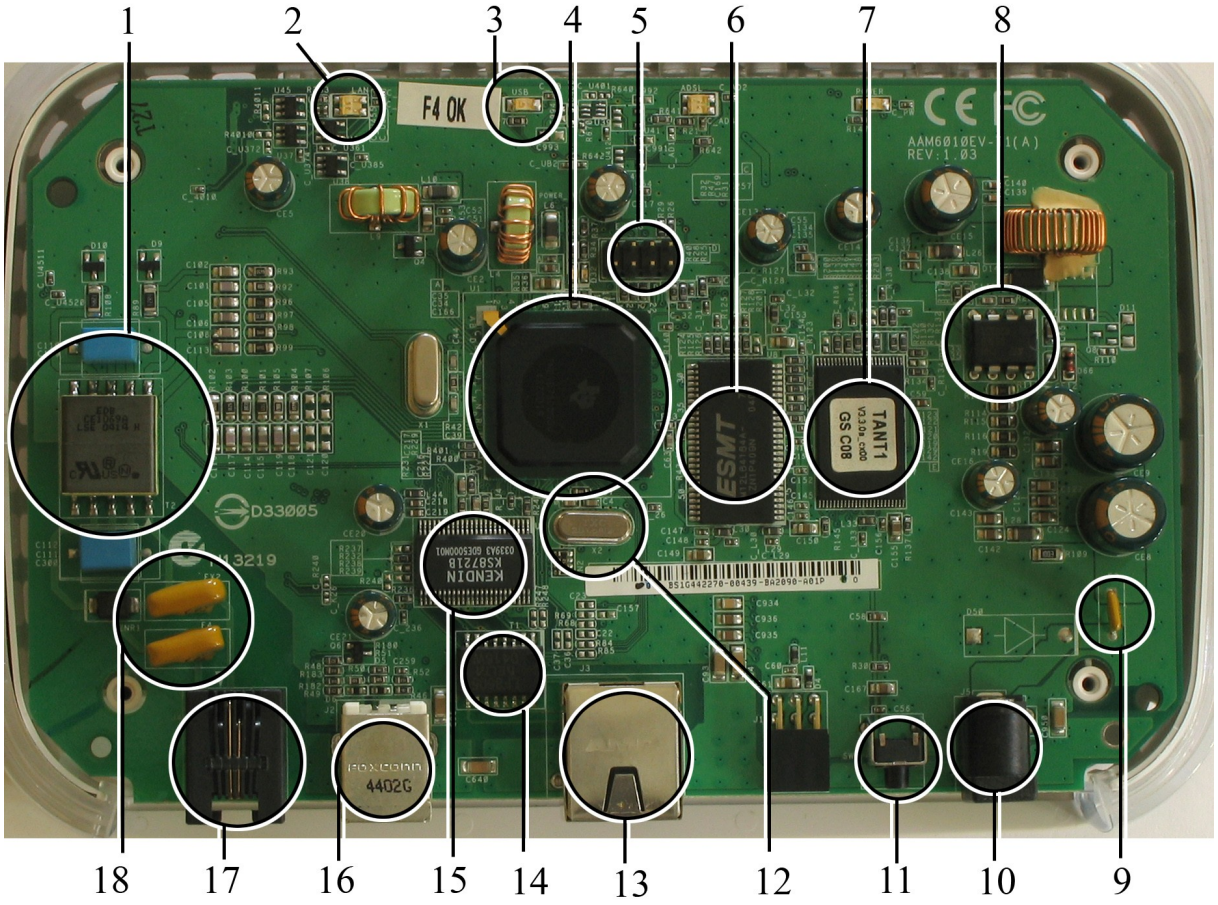
- CPU, RAM, ROM (mostly Flash)
- Peripheral devices to interface with outside world
  - Include debug/programming interfaces most of the time



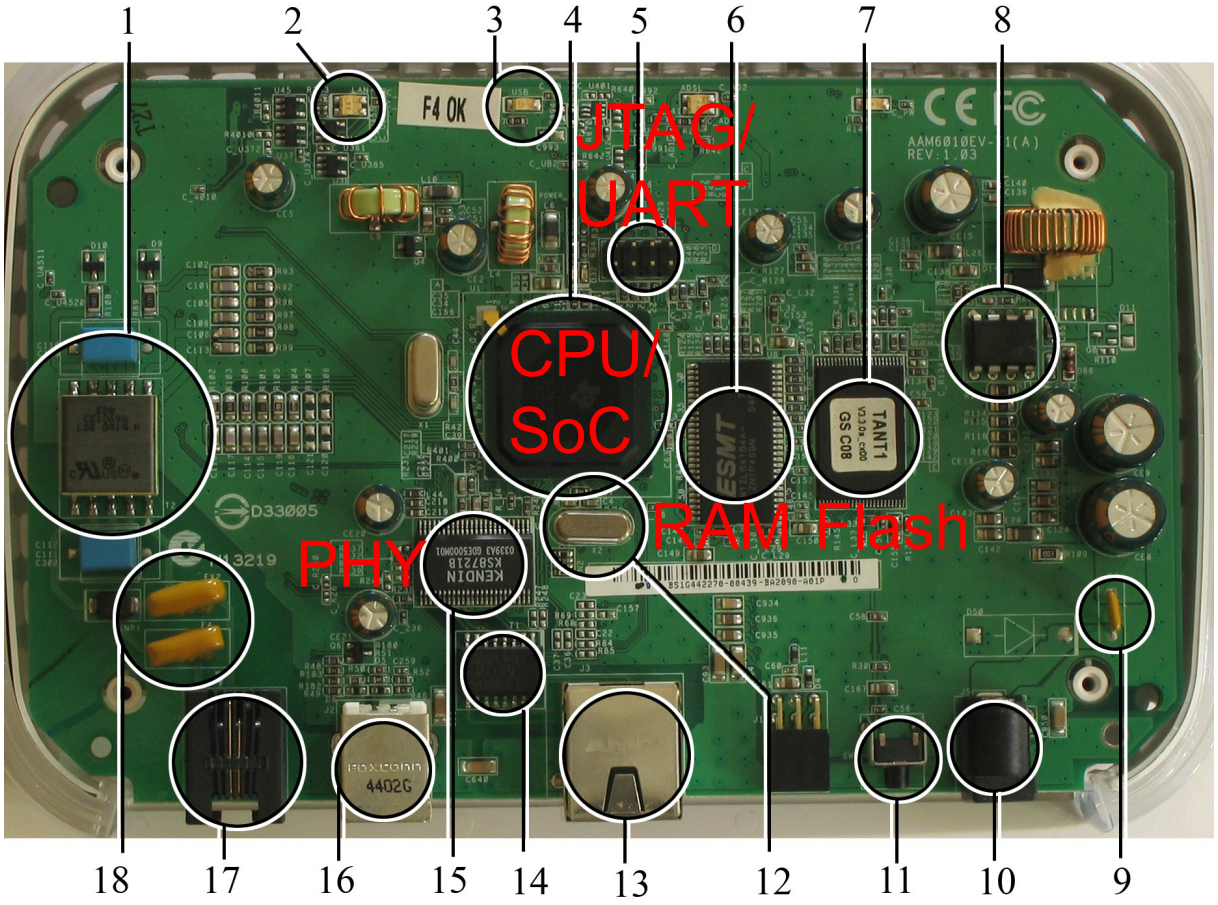
At least some of these components may also reside on chip (System On Chip - SoC)

- Power supply and glue logic
- Runs operating system or software application

A typical Embedded System



A typical Embedded System



Why is it so important ?

- Embedded systems are widespread
- They are often used for citical tasks:
  - Critical Infrastructures: SCADA, power grid, etc.
  - Mobility: car ECUs, aircraft control systems, etc.
  - Networking: Networking equipment, payment systems cell phones, etc.
  - ...
- Attacks can be disastrous

Why is it so important ?

- Embedded systems are widespread
- They are often used for citical tasks:
  - Critical Infrastructures: SCADA, power grid, etc.
  - Mobility: car ECUs, aircraft control systems, etc.
  - Networking: Networking equipment, payment systems cell phones, etc.
  - ...
- Attacks can be disastrous

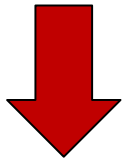
High security requirements

Why is it so important ?

- Embedded systems security analysis is:
  - Challenging
  - Not well supported
  - Time consuming (thus high costs)

Why is it so important ?

- Embedded systems security analysis is:
  - Challenging
  - Not well supported
  - Time consuming (thus high costs)



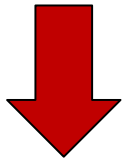
Divergence: high security requirements
vs.
available security analysis methods

Open vs. Closed Systems

- In comparison to PC system hardware, embedded systems are often “closed”:
  - Proprietary implementation
  - No in-depht documentation
  - Undocumented interfaces
  - Unknown communication protocols

Open vs. Closed Systems

- In comparison to PC system hardware, embedded systems are often “closed”:
  - Proprietary implementation
  - No in-depht documentation
  - Undocumented interfaces
  - Unknown communication protocols



Back to: “security-by-obscurity” ?

Open vs. Closed Systems

- High importance that embedded systems can be analyzed for security
- Tradeoff: high-level vs. low-level security analysis
  - Which vulnerabilities should be found ?
  - How much time should be invested ?
  - How much time is required at least ?

High-Level Analysis

- Idea:
  - We analyze communication protocols
  - Replay attacks ?
  - Fuzz Testing
  - High-level monitoring of embedded systems
    (i.e. does it crash, does it perform unintended tasks ?)
  - Easy to do without low-level access to system

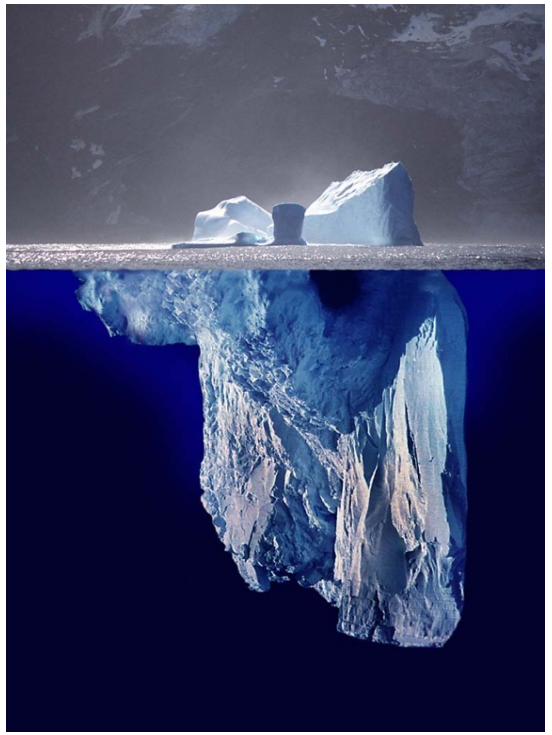
High-Level Analysis

- Idea:
  - We analyze communication protocols
  - Replay attacks ?
  - Fuzz Testing
  - High-level monitoring of embedded systems
    (i.e. does it crash, does it perform unintended tasks ?)
  - Easy to do without low-level access to system
- Drawback:
  - We don't know whether implementation is secure
  - Analysis is very limited, no insight into implementation

Low-Level Analysis

- Challenging and time-consuming
- Allows us to analyze implementation
- Very powerful
- Provides in-depht insight

- Answers questions:
  - Is implementation secure ?
  - Are there protection mechanisms ?
  - Do they work ?
  - How far can possible attacks reach ?
  - Etc.



Security Analysis / Attack Goals

- Full console access on device
  (e.g. gain root access on embedded system)
- Analyze software for bugs, software reverse engineering
  (e.g. find remote buffer overflows)
- Unlock restricted features
- Build alternative firmware or counterfeit products
- Extract secrets (e.g. encryption keys), etc.

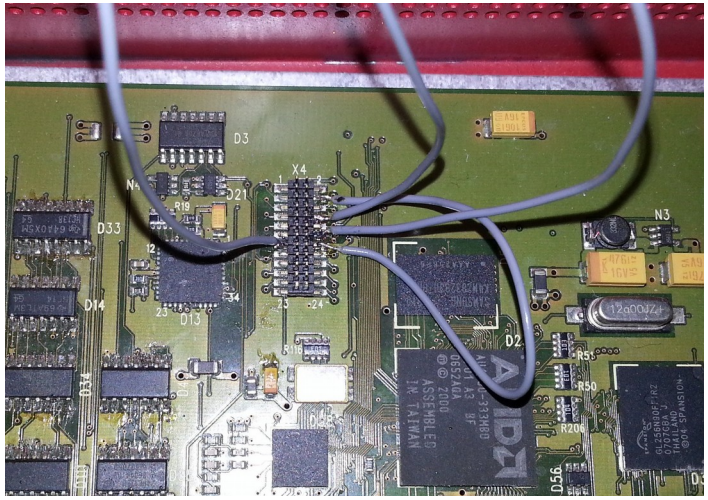
Analysis Techniques

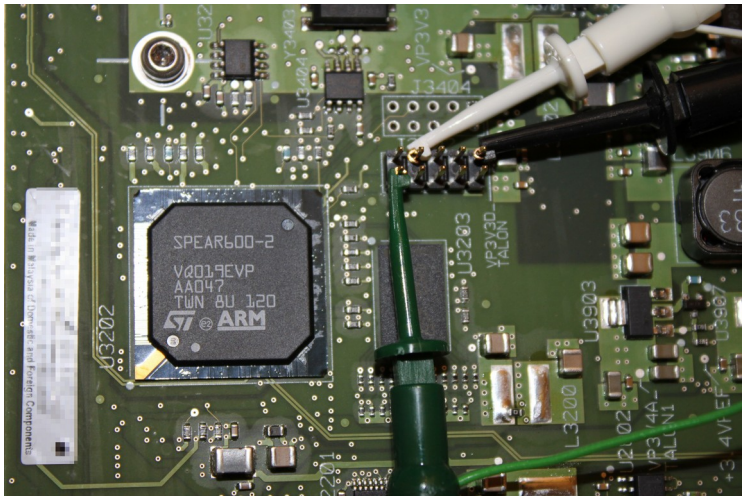
- How can we ?
  - Extract software
  - Debug system
  - Sniff / inject signals
  - Reverse engineer SW/HW design
  - ...

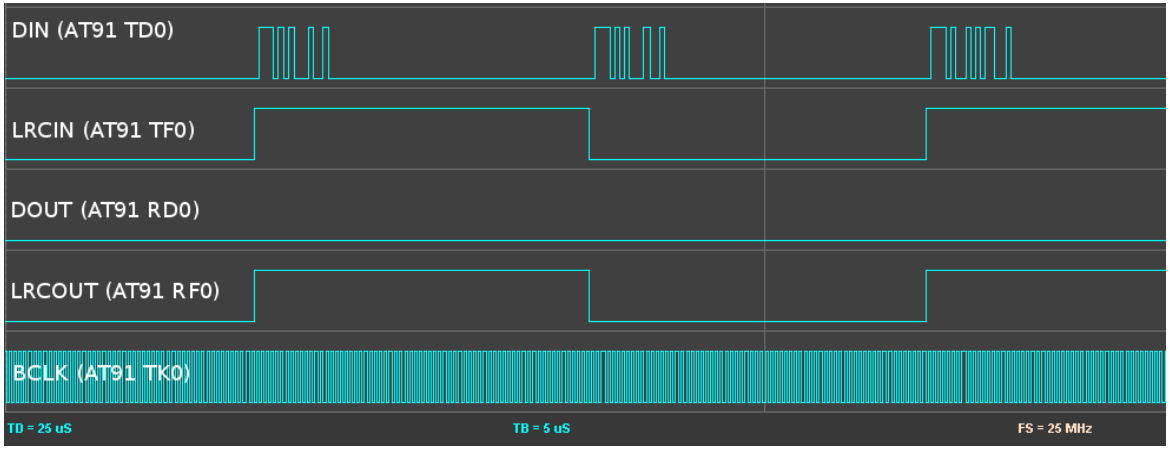
Analysis Techniques

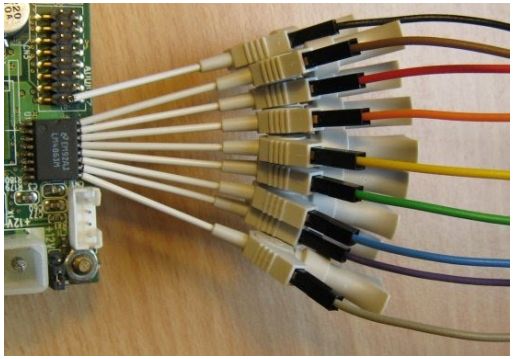
- Wide range of techniques available, depends on system and attack goals
- Monitor hardware components, override signals (oscilloscope, logic analyzer, waveform generator, ...)
- Monitor / modify device communication with environment, desolder components, etc.
- Use programming/debugger interfaces (UART, JTAG, ...)

Examples









Source: lynx-india.com

Firmware Extraction

- Firmware might reside in *external* or *internal* memory
- Internal memory more challenging, fault injection attacks might be helpful (next lecture)
- Possible ways to get to the firmware:
  - Desoldering
  - Downloadable firmware upgrades (e.g. Internet)
  - Logic sniffing
  - Console access
  - ...

Firmware Analysis

- Static anaylsis
  - Disassembly (typically non-x86 code, e.g. ARM, MIPS, etc.)
  - String analysis
  - Etc.
- Dynamic anaylsis
  - Requires debugging setup (can be difficult)
  - Emulation vs. real execution

Types of Embedded Systems

- Small systems:
  - Example: calculator, small control systems, etc.
  - Typically no OS
  - Strongly resource contrained
- Medium/Large systems:
  - Example: smart phone, network router
  - Typically run OS
  - Resource constrained

Embedded System Emulation

- Idea:
  - Emulate the embedded system
  - Example: qemu-system-mips
  - We run the implementation there
  - No resource constraints, full debug functionality, etc.
- Challenges:
  - Black-box peripherals can't be virtualized
  - Firmware/OS might not support emulator

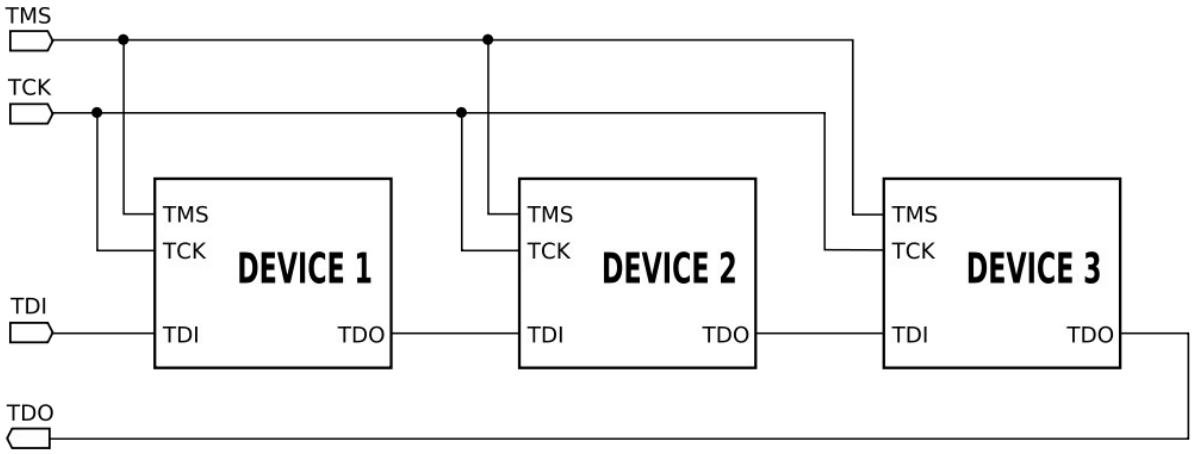
Embedded System Debugging

- Embedded systems often have debug and programming interfaces
- Necessary for production and manufacturing testing
- Often hidden on PCB

- Serial Console
- JTAG port (low-level access, proprietary extensions)

Joint Test Action Group (JTAG)

- Test Access Port (programming and debugging, e.g. with gdbserver) and Boundary-Scan (pin testing)
- Based on state-machine and (shift) registers, daisy chaining possible



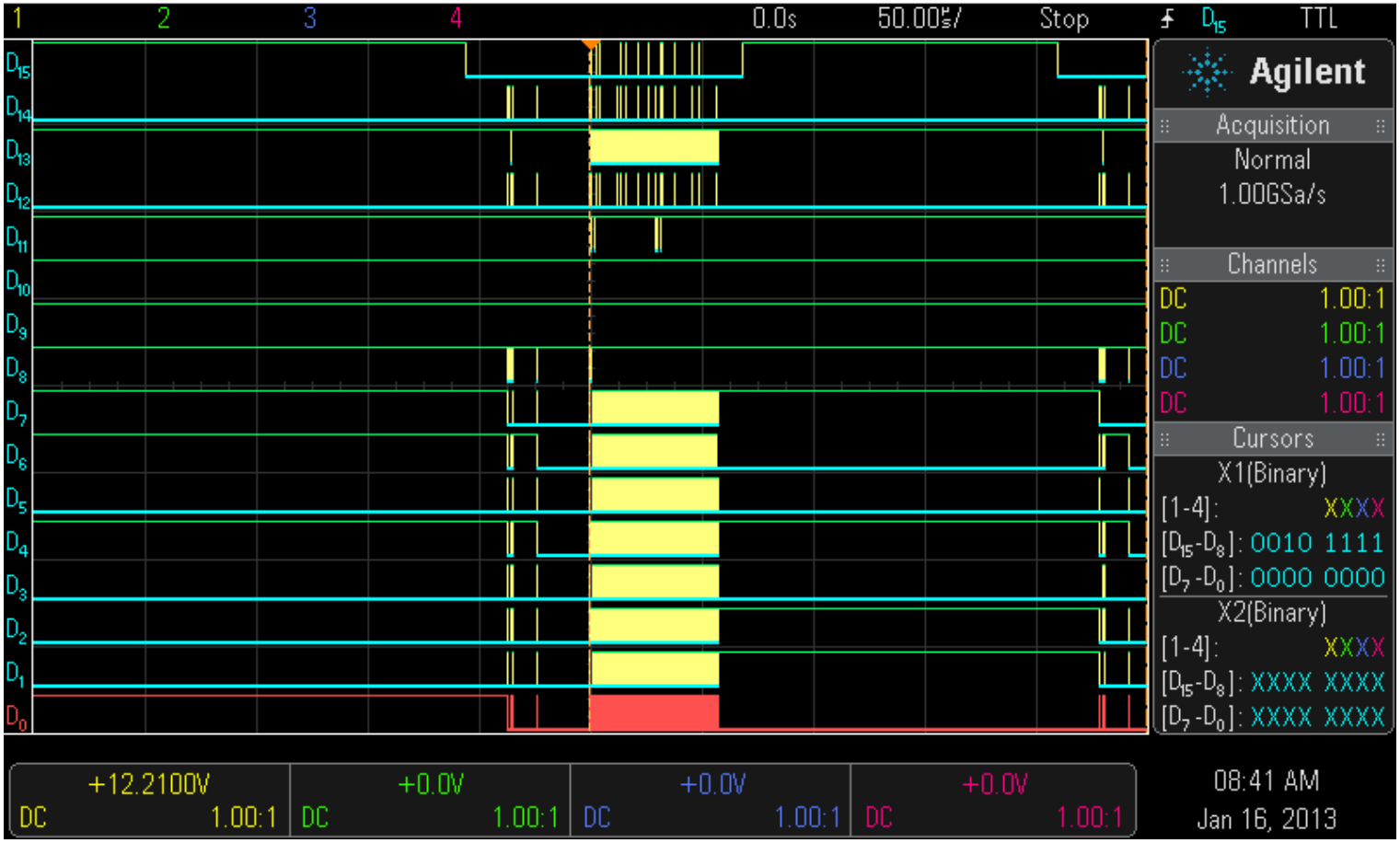
Using JTAG

- Different JTAG dongles for different controllers/architectures
- Can be attached to debug server (i.e. gdb-jtag-arm)
- Full debugging support
- Possible access to other devices in JTAG chain (e.g. memories)

Sniffing HW signals

- Attach logic analyzer or oscilloscope to PCB
- Capture signals
- Signal analysis on PC possible (e.g. Python script)
- Example: Communication between SD-Card controller and NAND Flash

Sniffing HW signals



Sniffing HW signals

- What do the signals mean ?
  - Find datasheets of (similar) components
  - What does the system do at the time of analysis ?
  - Is the bit-ordering correct ?
  - Bruteforce bit ordering possible

Sniffing: UART console discovery

- Use a scope to measure “suspicious” pins/traces
- Use datasheets to discover UART pins
  - Good candidates to test
- Reset the system
- At reset, bootloaders typically write to the console (e.g. Version string or bootloader identification)
- UART signal easily distinguishable

Signal Injection

- Signals can be injected as well
- Need to understand HW communication protocol
- Manual stimulus:
  - Microcontroller or FPGA
  - Used to generate signals
    (i.e. according to assumed communication protocol)
  - Scripting over PC (more diversity)

- Example:
  - Manual memory dump of SD-card NAND flash memory

Reverse Engineering

- Embedded software:
  - Use established reverse engineering approaches
  - I.e. disassembly, string analysis, etc.
- Embedded hardware:
  - Identify standard components
  - Find datasheets
  - Trace circuit lines
  - Measurements / signal sniffing
  - Allows HW reconstruction, but may be limited

Countermeasures

- Remove/obscure programming/debugging interfaces
- Hard to open enclosures, epoxy encapsulation, etc.
- SoC design
  (may also be smaller and cheaper for manufacturer)
- Tamper detection sensors, tamper response
  (e.g. reset system, delete flash memory, ...)
- ...

Newer Designs (1)

- Systems are getting smaller and include more security features
- SoC designs
- We can no longer easily access components / memory
- Programming / debug interfaces still exist, but there might be protections (e.g. passwords, encryption, etc.)
- Proprietary or secret debug ports and commands

Newer Designs (2)

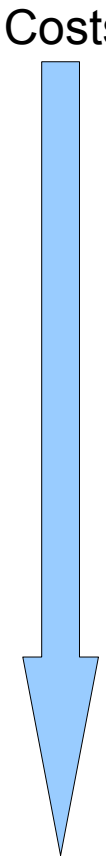
- Security relevant embedded devices are everywhere
- Manufacturers implement software or custom logic (e.g. on ASICs or FPGAs), but we can neither access nor analyze it ?
- Is this the end of security analysis on these devices ?
- Do we have to move back to security-by-obscurity and trust the manufacturer ?

Introduction to Implementation Attacks

Motivation

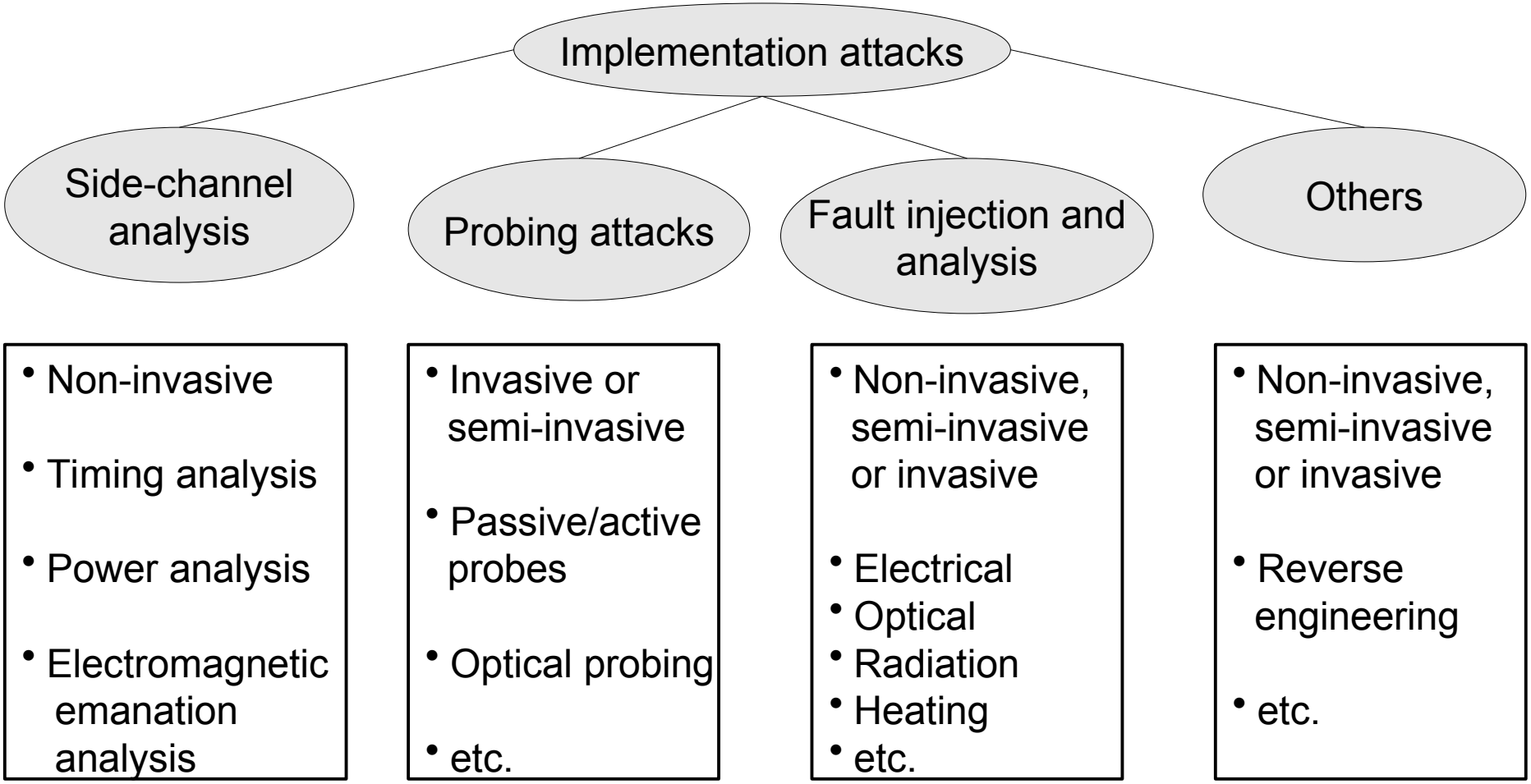
- If security design goes down to chip level, we need to do that as well
- Integrated circuits (ICs) can be analyzed and tampered with
- IC security measures can always be bypassed, but with increasing effort (time & money)

Type of IC Access



- Non-invasive
  - We don't have to open the chip
- Semi-invasive
  - The chip has to be decapsulated, so that die is visible
  - Passivation layer stays intact
- Invasive
  - The chip is fully decapsulated
  - Passivation layer is (partially) removed
  - Physical contact to chip signals possible

Taxonomy


Implementation attacks
Side-channel analysis
Probing attacks
Fault injection and analysis
Others
• Non-invasive
• Timing analysis
• Power analysis
• Electromagnetic emanation analysis
• Invasive or semi-invasive
• Passive/active probes
• Optical probing
• etc.
• Non-invasive, semi-invasive or invasive
• Electrical
• Optical
• Radiation
• Heating
• etc.
• Non-invasive, semi-invasive or invasive
• Reverse engineering
• etc.

Side Channel Analysis

- A system may leak security relevant side channel information
- Many side-channels (timing, power, electromagnetic emanation, acoustic or optical emanation, heat, etc.)
- By analyzing this information, we might be able to learn sensitive information (e.g. encryption keys, passwords, etc.)

Timing Analysis

- We interact with the system and closely monitor the timing during operation
- For example, we might monitor the timing of a password check
  - Start timer when guessed password is sent to device
  - Stop timer when response (i.e. 'wrong password') is received
  - Compare time measurements for different guesses
  - See if we can draw a conclusion and determine the correct password

A bad password check (1)

```
bool check_password(char *passwd)
{
        for (int i=0; i<pass_len; i++)
        {
                if (passwd[i] != stored_passwd[i])
                        return false;
        }

        return true;
}
```

A bad password check (2)

Terminates as soon a byte is wrong

```
bool check_password(char *passwd)
{
        for (int i=0; i<pass_len; i++)
        {
                if (passwd[i] != stored_passwd[i])
                        return false;
        }

        return true;
}
```

Based on timing information, we can easily guess the password

A better password check (1)

```
bool check_password(char *passwd)
{
        int err=0;
        for (int i=0; i<pass_len; i++)
        {
                err |= passwd[i] ^ stored_passwd[i];
        }

        if (err != 0)
                return false;
        return true;
}
```

A better password check (2)

```
bool check_password(char *passwd)
{
        int err=0;
        for (int i=0; i<pass_len; i++)
        {
                err |= passwd[i] ^ stored_passwd[i];
        }

        if (err != 0)
                return false;
        return true;
}
```

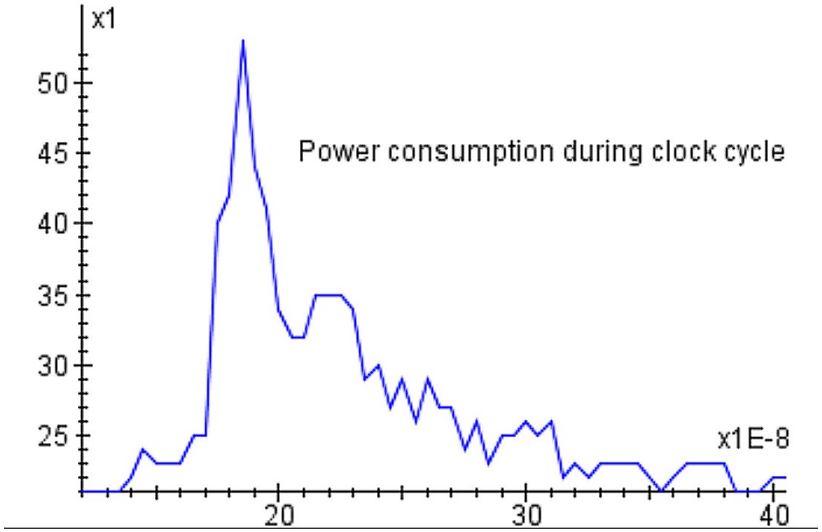
Constant time

Simple Power Analysis (1)

- The power consumption of a processor depends on the instruction executed
- We closely monitor the power consumption during clock cycles (i.e. time domain)
- For a given instruction, the power consumption also depends on the data processed

Simple Power Analysis (2)

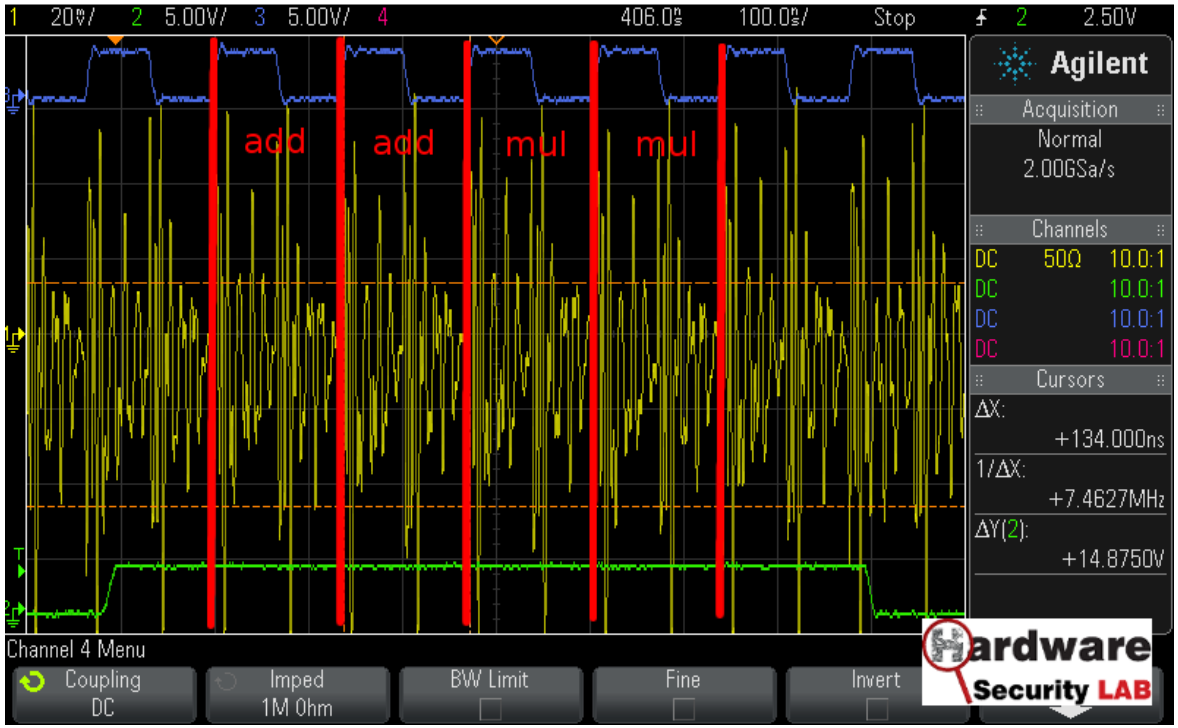
- Example:



Source: riscure.com

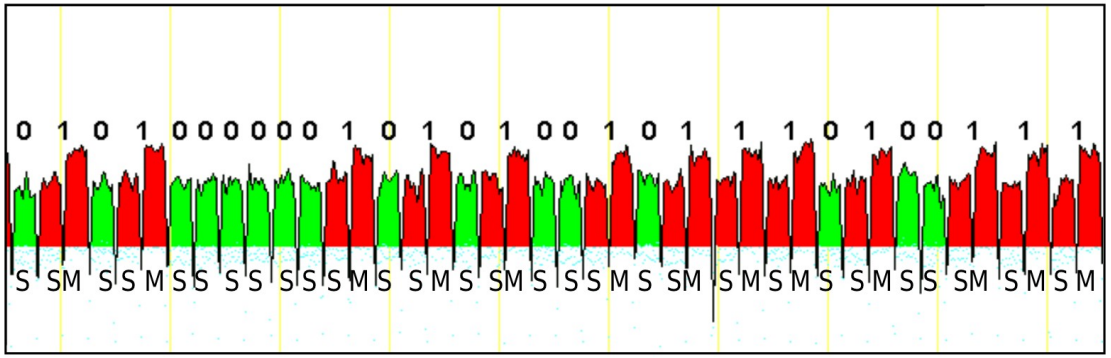
- By analyzing the power consumption, it might be possible to determine the *instruction* and *data*

Simple Power Analysis (3)



Vulnerable RSA exponentiation

- Example:



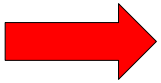
Source: 'Protecting FPGAs from Power Analysis', Cryptography Research Inc.

- Using SPA, we can complete recover the RSA secret key in this case !

- Drawback: We need a good S/N ratio for SPA to work, can be increased through averaging over multiple measurements

Differential Power Analysis (1)

- More powerful than SPA
- Combines many measurements, considers only *data*
- “Oracle”: Uses power model (Hamming weight or distance)

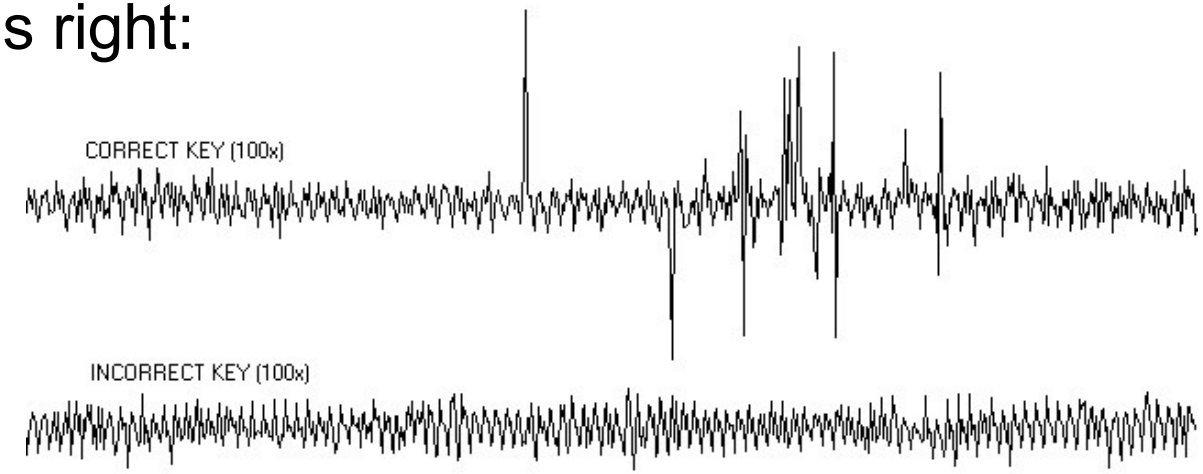


Implementation needs to be known
(but it can be guessed from power traces as well – e.g. AES) !

Differential Power Analysis (2)

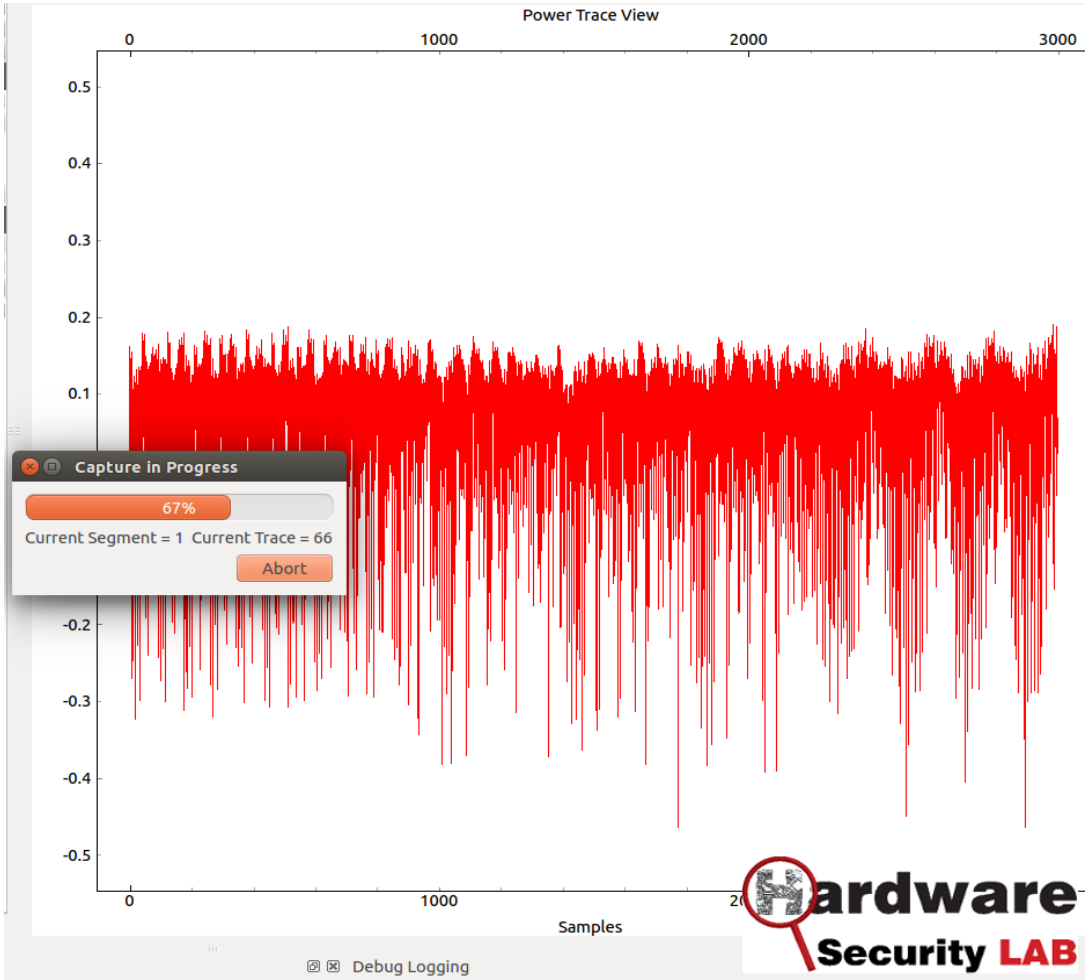
- Calculates statistical correlation between model and measurements, thus also known as CPA (correlation power analysis)

- DPA curve shows peaks if guessed value (e.g. key) was right:

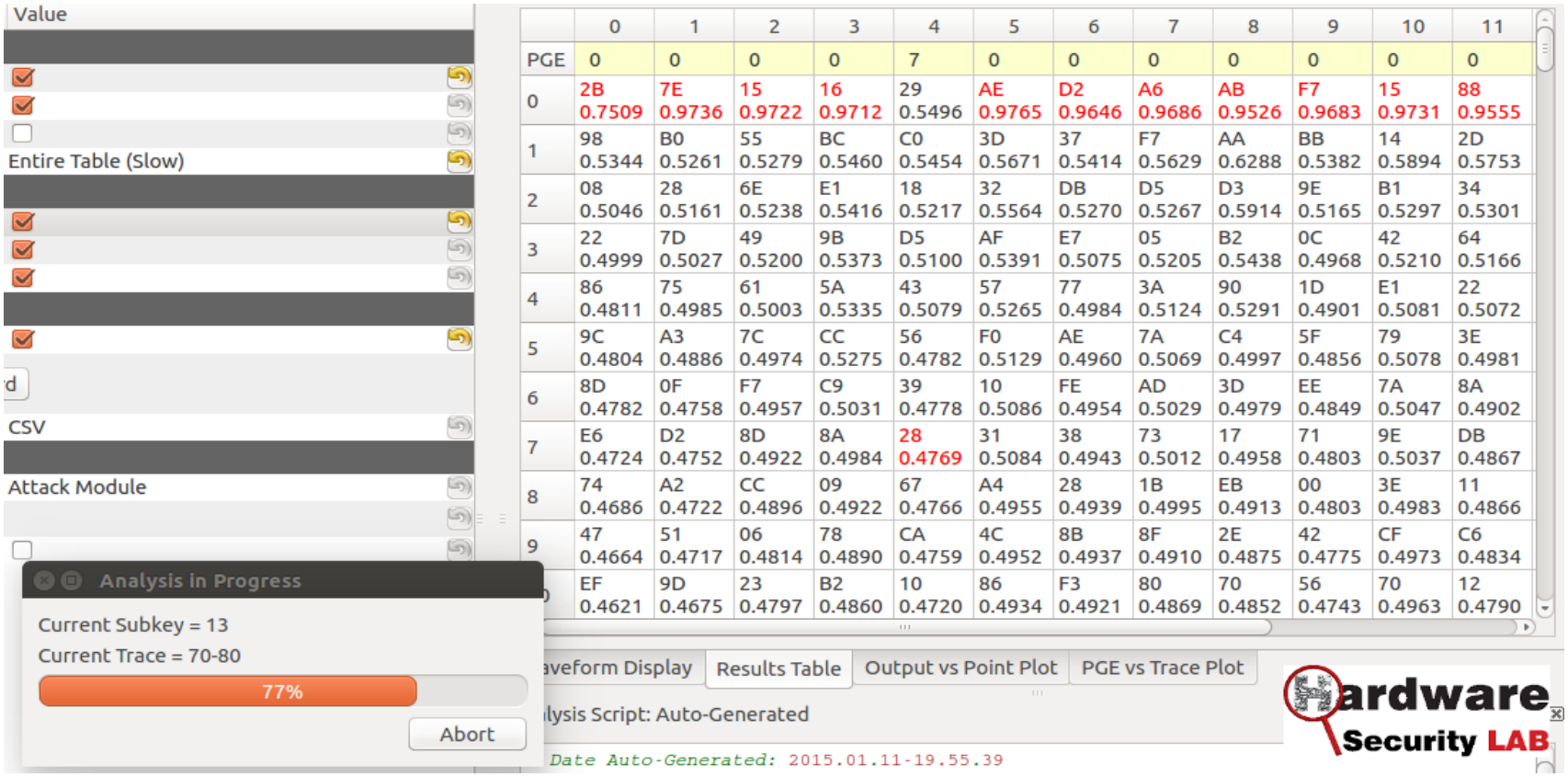


Source: 'Protecting FPGAs from Power Analysis', Cryptography Research Inc.

Differential Power Analysis (3)



Differential Power Analysis (4)



	0	1	2	3	4	5	6	7	8	9	10	11
PGE	0	0	0	0	7	0	0	0	0	0	0	0
0	2B 0.7509	7E 0.9736	15 0.9722	16 0.9712	29 0.5496	AE 0.9765	D2 0.9646	A6 0.9686	AB 0.9526	F7 0.9683	15 0.9731	88 0.9555
1	98 0.5344	B0 0.5261	55 0.5279	BC 0.5460	C0 0.5454	3D 0.5671	37 0.5414	F7 0.5629	AA 0.6288	BB 0.5382	14 0.5894	2D 0.5753
2	08 0.5046	28 0.5161	6E 0.5238	E1 0.5416	18 0.5217	32 0.5564	DB 0.5270	D5 0.5267	D3 0.5914	9E 0.5165	B1 0.5297	34 0.5301
3	22 0.4999	7D 0.5027	49 0.5200	9B 0.5373	D5 0.5100	AF 0.5391	E7 0.5075	05 0.5205	B2 0.5438	0C 0.4968	42 0.5210	64 0.5166
4	86 0.4811	75 0.4985	61 0.5003	5A 0.5335	43 0.5079	57 0.5265	77 0.4984	3A 0.5124	90 0.5291	1D 0.4901	E1 0.5081	22 0.5072
5	9C 0.4804	A3 0.4886	7C 0.4974	CC 0.5275	56 0.4782	F0 0.5129	AE 0.4960	7A 0.5069	C4 0.4997	5F 0.4856	79 0.5078	3E 0.4981
6	8D 0.4782	0F 0.4758	F7 0.4957	C9 0.5031	39 0.4778	10 0.5086	FE 0.4954	AD 0.5029	3D 0.4979	EE 0.4849	7A 0.5047	8A 0.4902
7	E6 0.4724	D2 0.4752	8D 0.4922	8A 0.4984	28 0.4769	31 0.5084	38 0.4943	73 0.5012	17 0.4958	71 0.4803	9E 0.5037	DB 0.4867
8	74 0.4686	A2 0.4722	CC 0.4896	09 0.4922	67 0.4766	A4 0.4955	28 0.4939	1B 0.4995	EB 0.4913	00 0.4803	3E 0.4983	11 0.4866
9	47 0.4664	51 0.4717	06 0.4814	78 0.4890	CA 0.4759	4C 0.4952	8B 0.4937	8F 0.4910	2E 0.4875	42 0.4775	CF 0.4973	C6 0.4834
	EF 0.4621	9D 0.4675	23 0.4797	B2 0.4860	10 0.4720	86 0.4934	F3 0.4921	80 0.4869	70 0.4852	56 0.4743	70 0.4963	12 0.4790

Password check revisited

```
bool check_password(char *passwd)
{
        int err=0;
        for (int i=0; i<pass_len; i++)
        {
                err |= passwd[i] ^ stored_passwd[i];
        }

        if (err != 0)
                return false;
        return true;
}
```

Power consumption depends on data !

Based on power analysis, we can guess the password

SCA Countermeasures

- Leakage reduction (e.g. through balancing)
- Noise introduction
- Masking (e.g. through insertion of random dummy cycles)
- Obfuscation
- etc.

Fault Injection and Analysis

- We intentionally inject faults into the IC
- Attempt to change normal device operation to the attackers advantage (e.g. to bypass password check, recover encryption keys, etc.)
- Transient and permanent faults

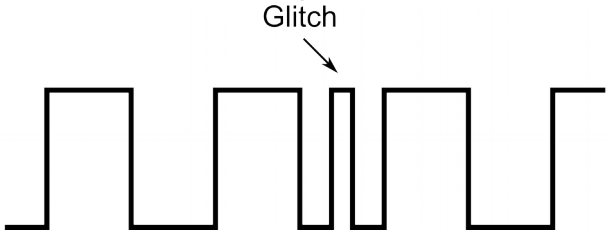
Electrical Fault Injection

- We inject a fault into the system by modifying the electrical operating environment of the IC

- Examples:
  - Clock glitching
  - Voltage glitching
  - Laser glitching

Clock Glitching

- For a short time, the IC receives a clock pulse that is too fast for the IC to fully process



- Some of the IC operations will work as intended (e.g. increase program counter), others will not finish and get interrupted

- Can be used to skip code (e.g. conditional jump in password check)

Bypass Password check (CLK glitch)

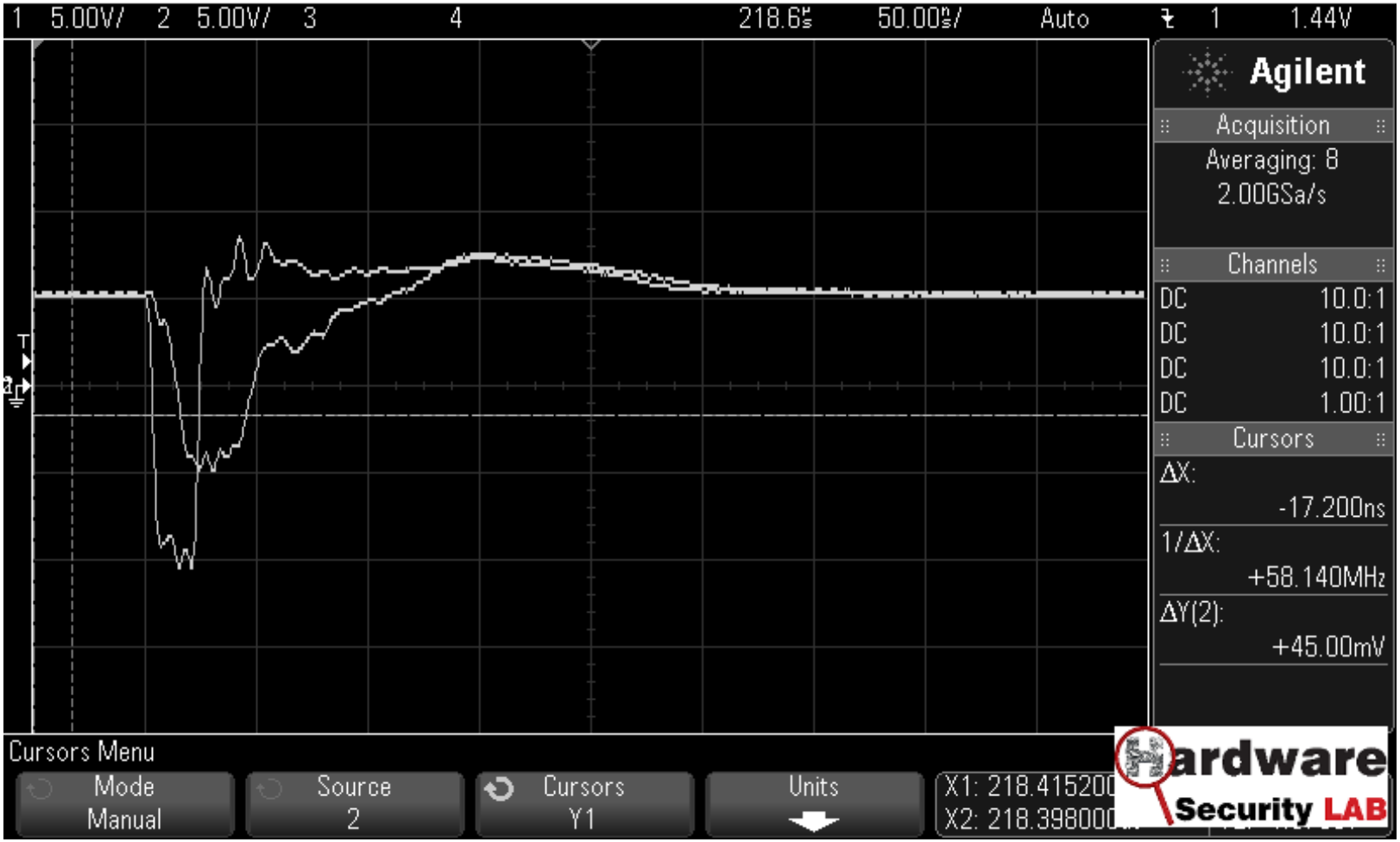
```
bool check_password(char *passwd)
{
        int err=0;
        for (int i=0; i<pass_len; i++)
        {
                err |= passwd[i] ^ stored_passwd[i];
        }

        if (err != 0)
                return false;
        return true;
}
```

Use CLK glitch to skip this code fragment

Voltage Glitching

- Voltage glitching: We supply the IC with a voltage out of normal operating range for a short period of time
- Depending on memory technology (e.g. Flash, SRAM), the stored charge in the memory cell is compared to a reference voltage
- Through the glitch, we might be able to change the reference voltage and thus make the IC read an arbitrary bit value

Voltage Glitch Example



Bypass Password check (Vcc glitch)

```
bool check_password(char *passwd)
{
        int err=0;
        for (int i=0; i<pass_len; i++)
        {
                err |= passwd[i] ^ stored_passwd[i];
        }

        if (err != 0)
                return false;
        return true;
}
```

Use Vcc glitch so that err is always read as '0' from SRAM

Fault Injection Countermeasures

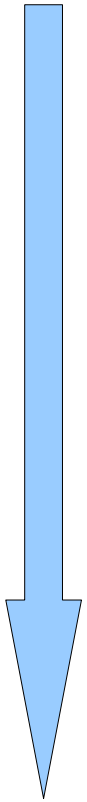
- Environmental sensors
- Tamper sensors
- Internal filtering
- Shielding
- etc.

Motivation for (semi) invasive attacks

- Non-invasive attacks
  - Limitations and countermeasures
  - Firmware might be locked (e.g., security fuse) and immune to glitch attacks
  - We have no insight into the hardware and its security features
- We can obtain more information by opening up the IC
  - Attack localization
  - Reverse engineering
  - Probing & signal Injection

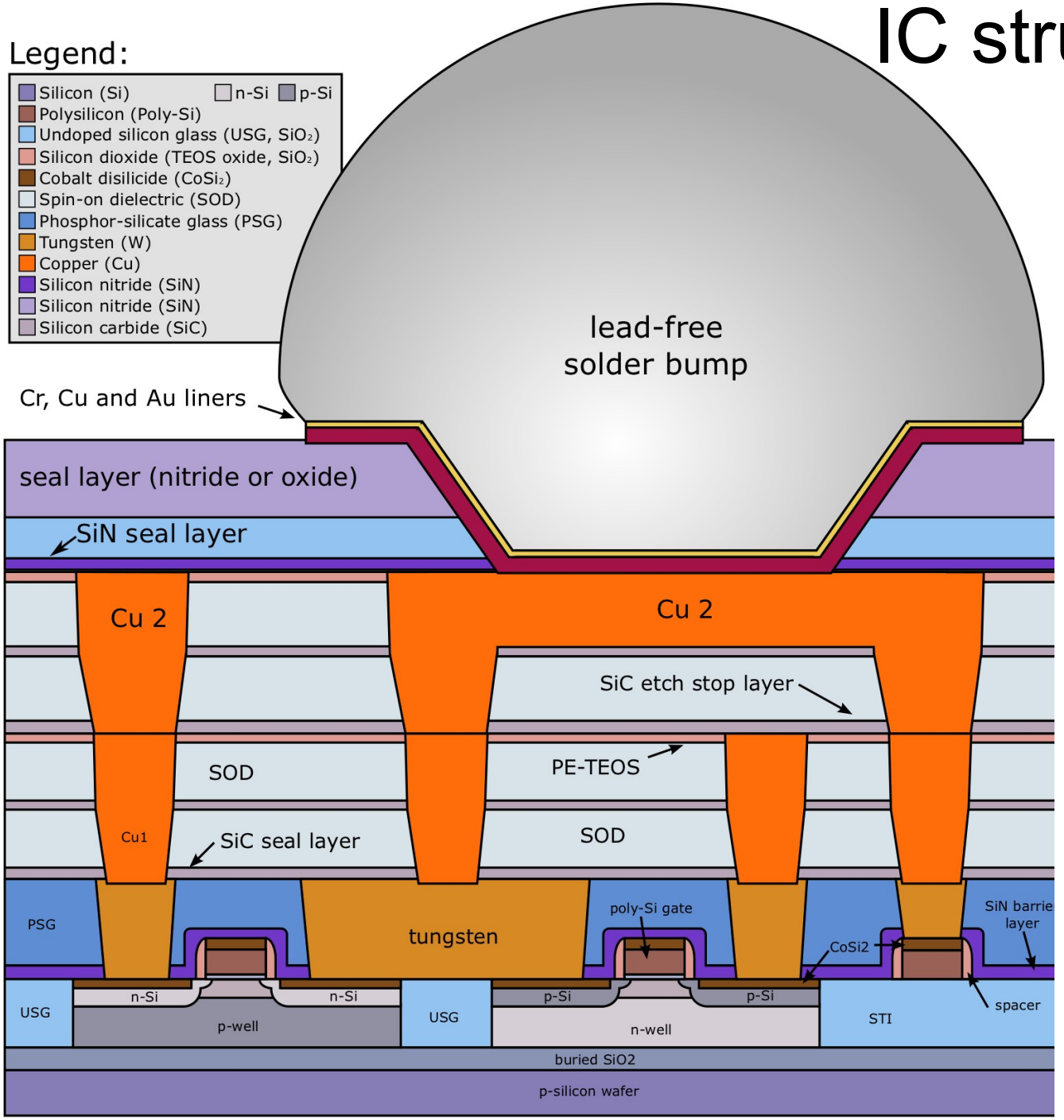
Type of IC Access

Costs



- Non-invasive
  - We don't have to open the chip
- Semi-invasive
  - The chip has to be decapsulated, so that die is visible
  - Passivation layer stays intact
- Invasive
  - The chip is fully decapsulated
  - At least the passivation layer is (partially) removed
  - Physical contact to chip signals possible

IC structure

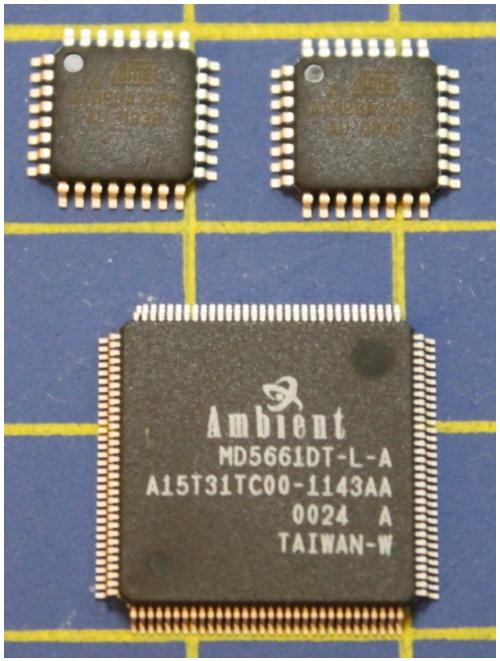


IC Decapsulation

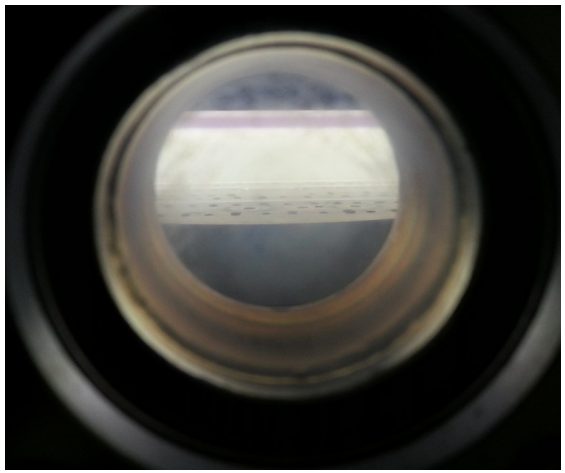
- Usually, die is covered in Epoxy package
- For semi-invasive and invasive attacks, we want to expose the die without destroying the IC
- Depending on attack type:
  - IC needs to stay fully operational
  - Bonding wires and pins might need to stay intact
- Multiple possibilities (wet/dry etching, Laser decapsulation, mechanical, etc.)

Wet Chemical Decapsulation

- Epoxy package is very resistant
- Epoxy can be dissolved in concentrated and heated up acids (usually fuming HNO_3, H_2SO_4 or a combination thereof)
- Bonding wires, pads and passivation layer stays intact, copper wires can be an issue
- Easy to conduct, but safety equipment necessary



Dry Etching Decapsulation



- Idea: We use a process gas mixture (e.g. Ar/CF4/O2) to ignite a plasma
- Reactive plasma acts both chemically and mechanically:
  - Free radicals (e.g. F, O)
  - Sputtering of ions
- Epoxy package can be dissolved
- Advantage: Very clean and gentle to die if done right, works with copper bonding wires
- Disadvantage: Expensive, might take long time

Laser Decapsulation

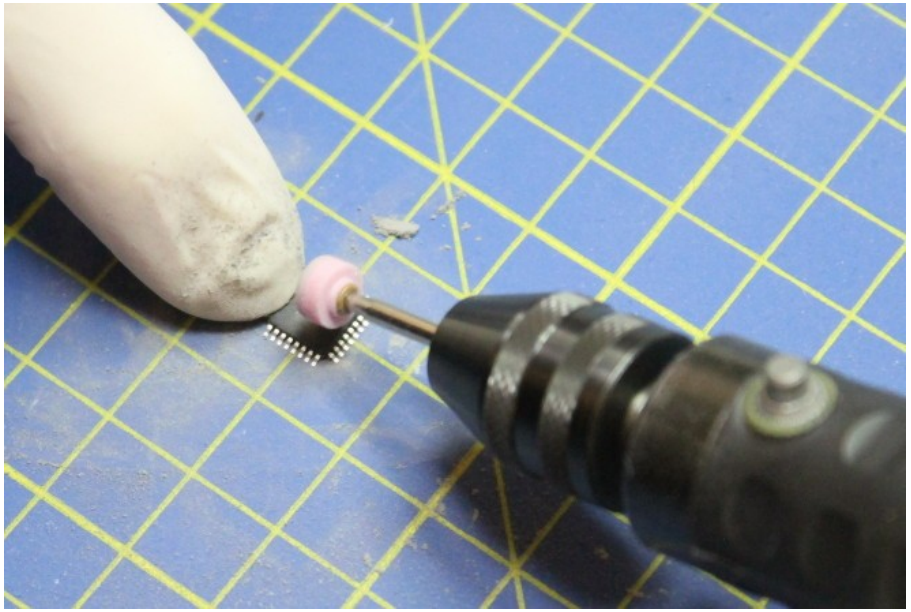
- Idea: Use CO2 Laser (IR, 10µm wavelength) to evaporate Epoxy
- Advantage: Works with copper wires due to heat transfer, Laser machines available at low cost (e.g., eBay), frequently found in hackerspaces
- Disadvantage: Can cause damage to the die !

Mechanical Decapsulation

- Very easy to conduct
- Fixtures exist (e.g., sharp knife in vise)
- Grinding (e.g., Dremel tool)

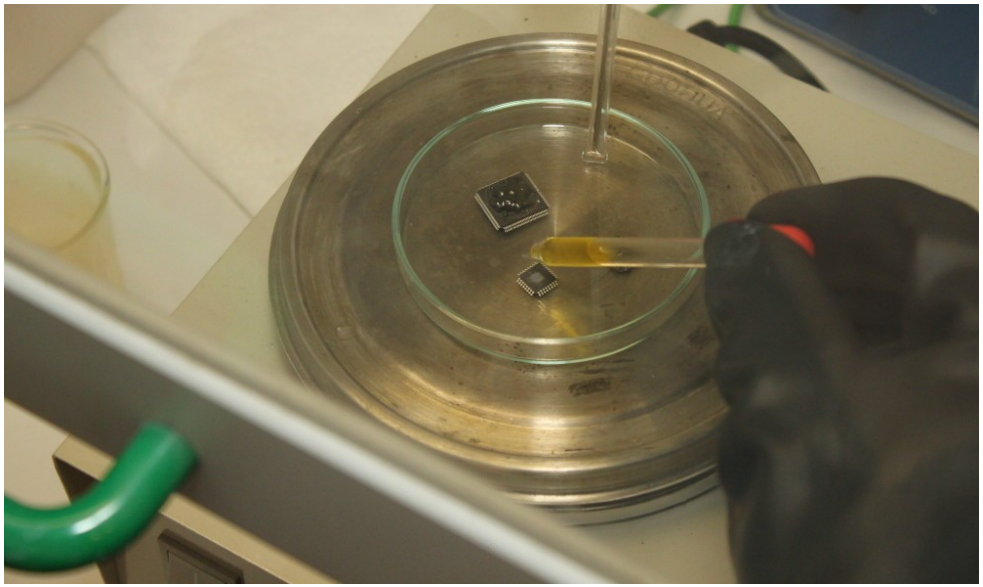
- Disadvantage: Only applicable for very old chips and/or for backside preparation

Our Process - Chemical Decapsulation (1)



1 – Carefully mill a cavity

Our Process - Chemical Decapsulation (2)



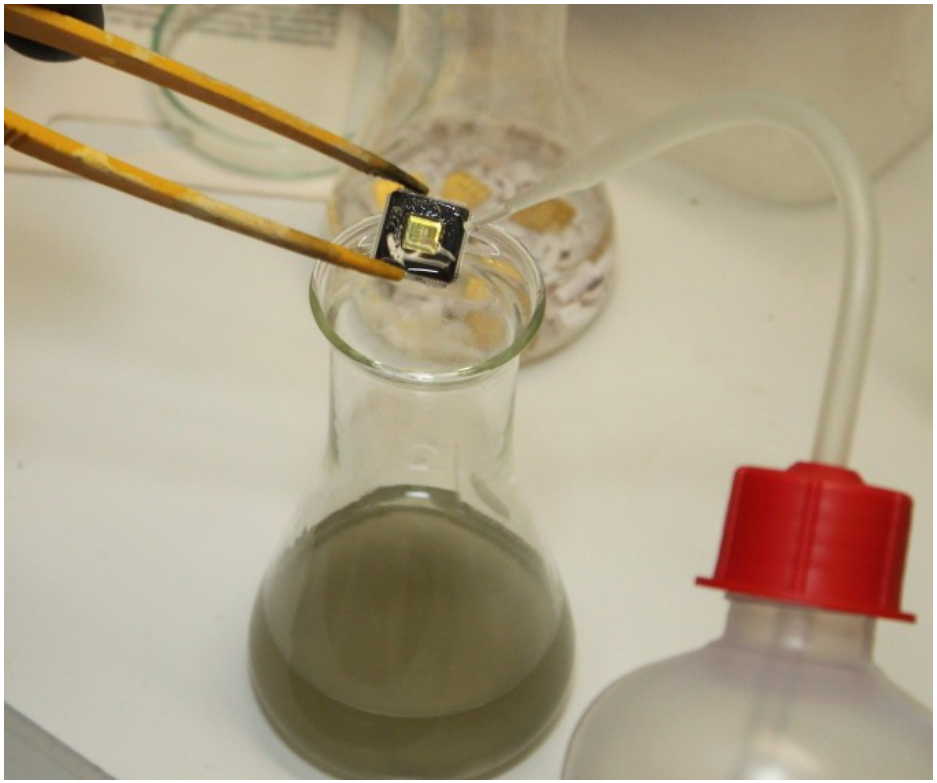
2 – Carefully apply nitric acid (HNO_3) / sulfuric acid (H_2SO_4) on hot plate

Safety equipment & fume hood

Our Process - Chemical Decapsulation (3)

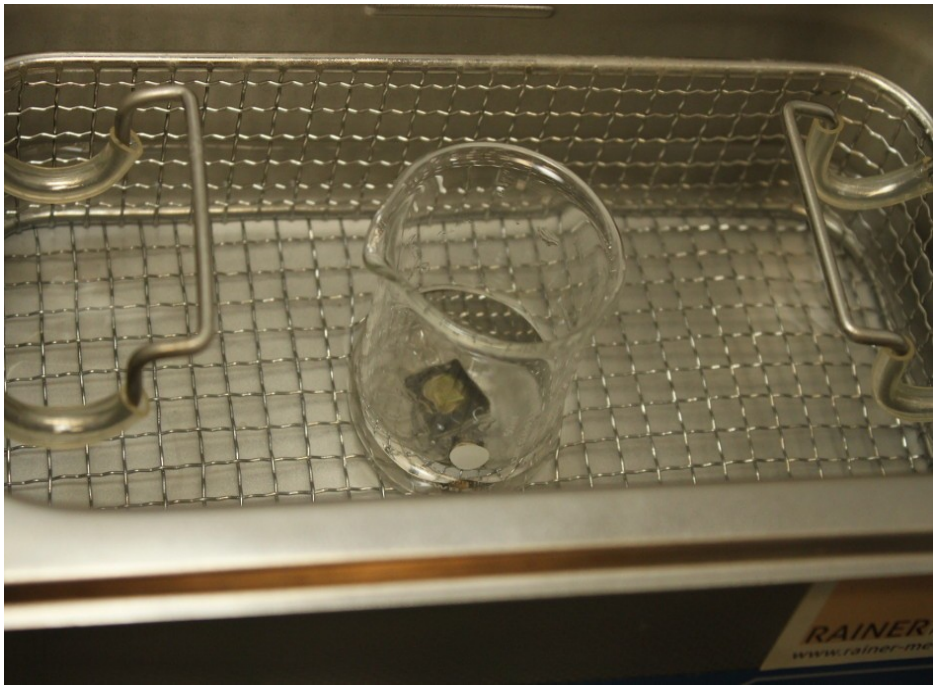
3 – Rinse in Acetone

Our Process - Chemical Decapsulation (4)



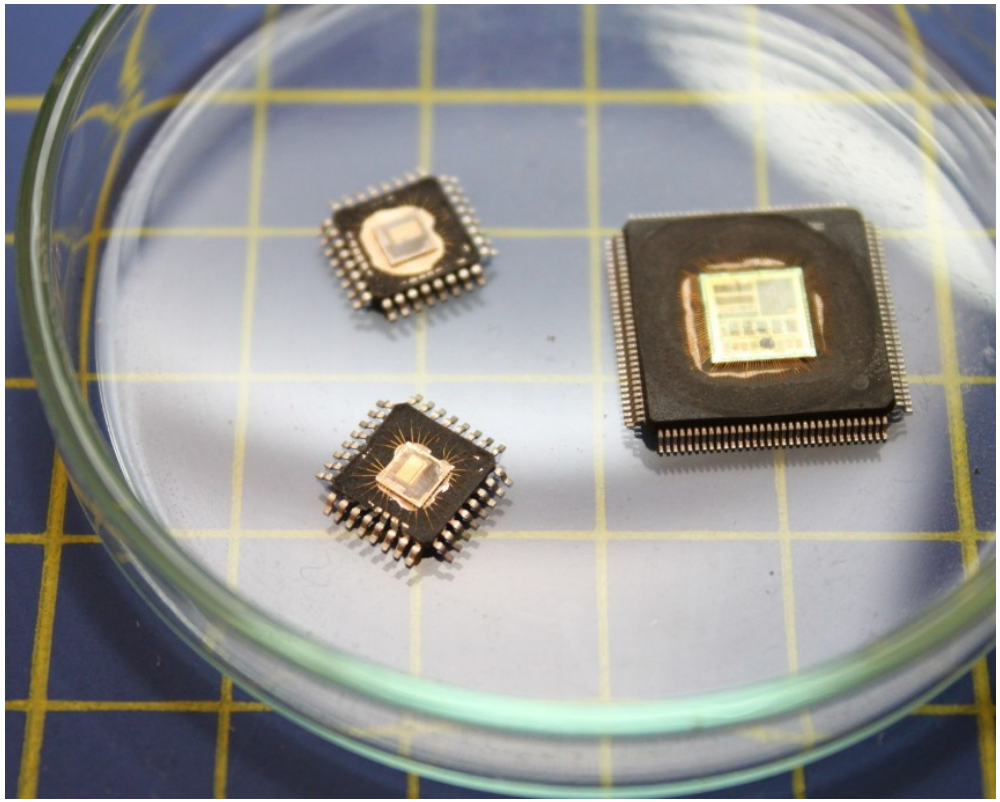
4 – Repeat etch & rinse until die fully exposed

Our Process - Chemical Decapsulation (5)



5 – Clean in Acetone in ultrasonic cleaner to remove remaining residue

Our Process - Chemical Decapsulation (6)



6 – Chip ready for further analysis and/or attack

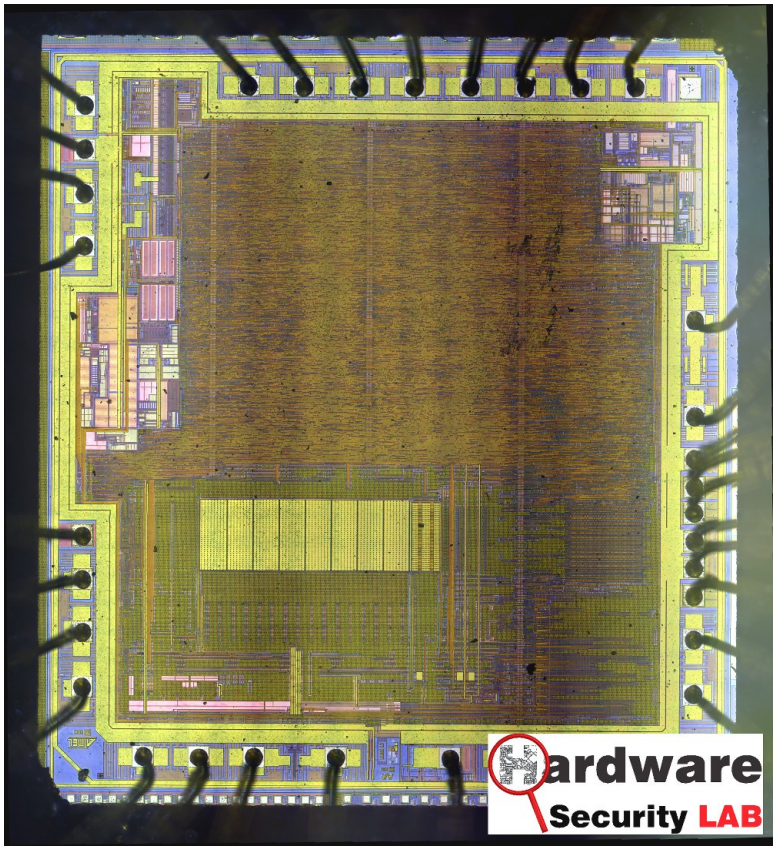
Chip is still functional

Overview: Semi-invasive Attacks

- Reverse Engineering (limited extent)
- Probing Attacks (limited extent)
- Laser Fault Injection
  - Backside attacks – use Near IR laser (Si is transparent to NIR)
  - Frontside attacks – Visible laser light, but metal obstructs view
- Photon Emission Analysis
- Voltage Contrast Imaging
- Localized Side Channel Attacks

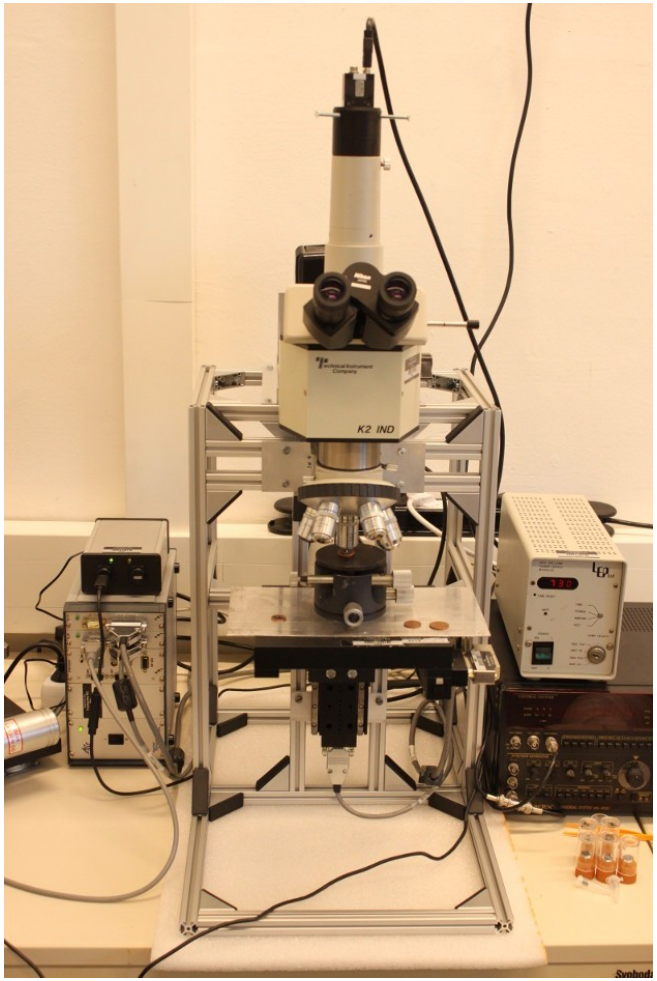
Optical Microscopy

- Useful to get an overview
- Optical resolution is limited
- Low magnification (<1000x)
- Low deptht of focus



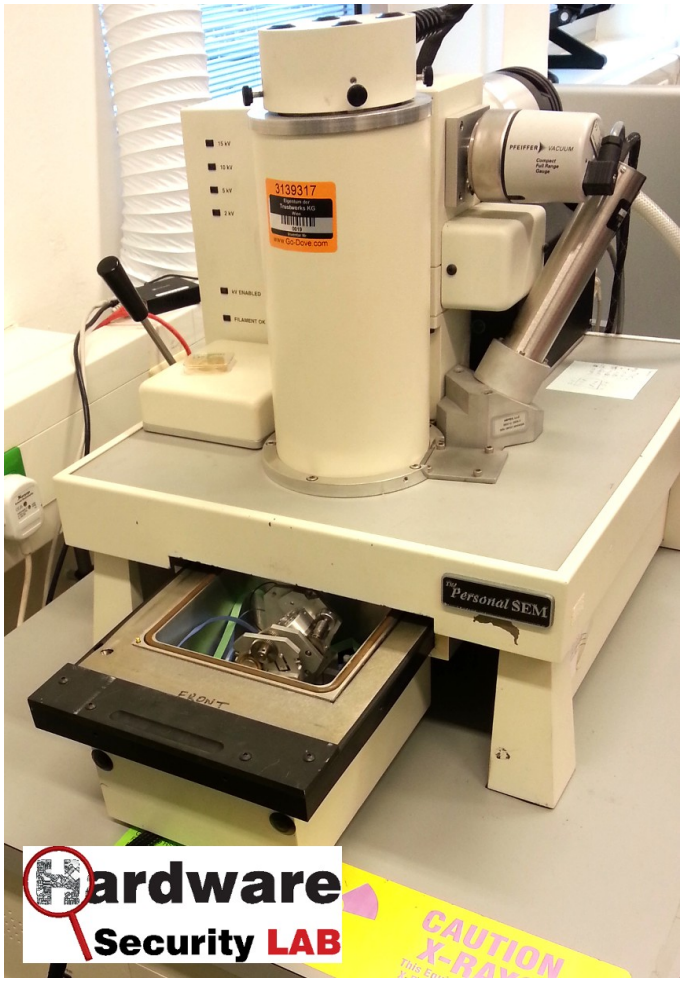
Optical Microscopy Improvements

- Motorized stage
- High resolution & high sensitivity digital camera
- Software to automatically take tiles
- Specialized stitching software to obtain high resolution die images
- Confocal scan head (changes color of objects with different focus)



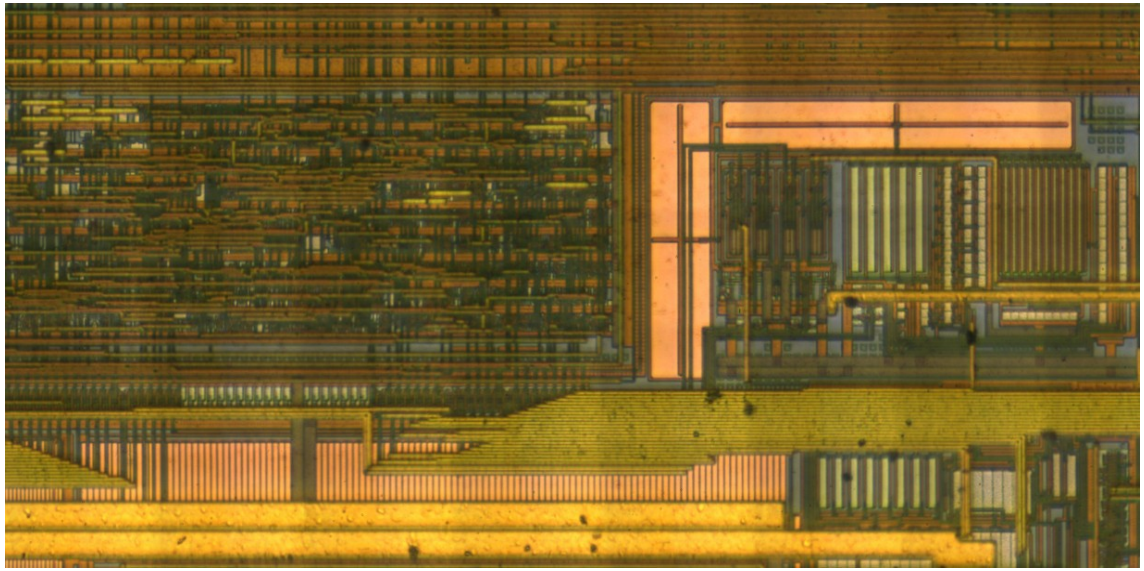
Scanning Electron Microscope

- Very versatile tool, operates under high vacuum
- Increasingly affordable (e.g., second hand)
- High depth of focus
- High magnification
- Disadvantage:
  - Electrical charging
  - Long scan times



Semi-invasive Reverse Engineering (1)

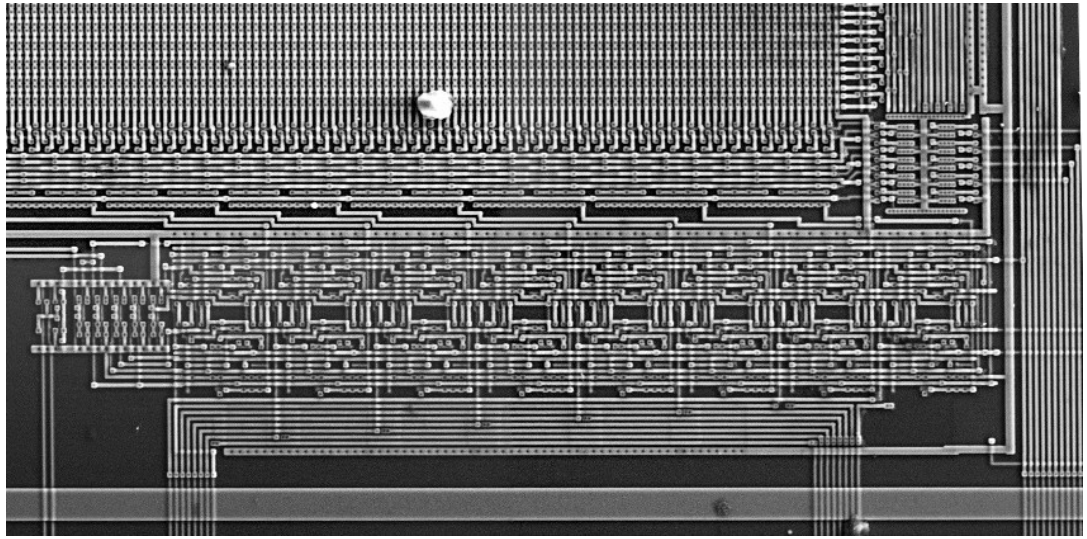
- In general you can only see top metal
- Limited insight



- Usable to identify probe pads, potential busses for probing, etc.

Semi-invasive Reverse Engineering (2)

- Older chips: visibility on logic as well

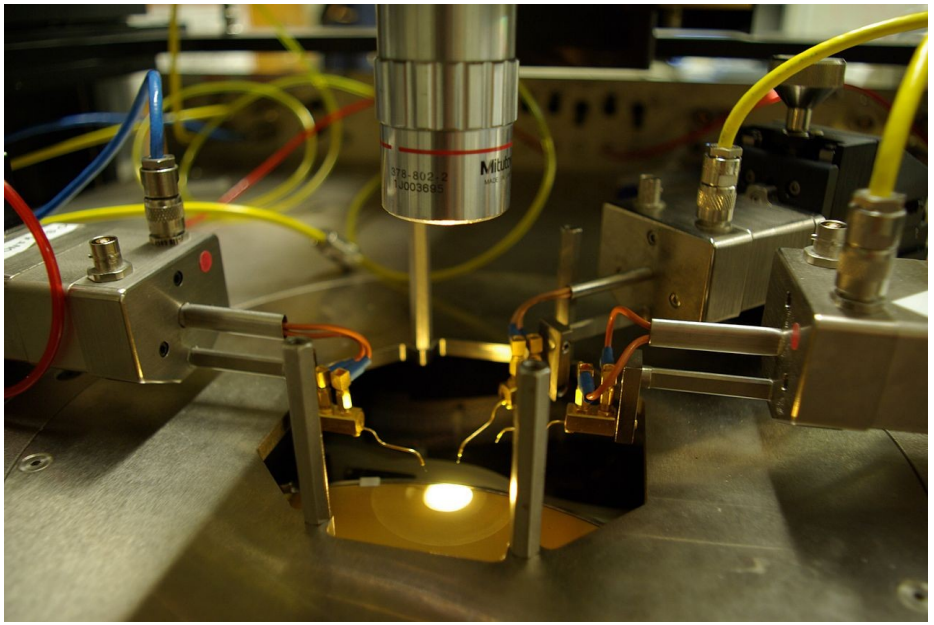


- We mostly reverse engineered an entire cryptographic chip this way:

Markus Kammerstetter, Markus Muellner, Daniel Burian, Christian Platzer and Wolfgang Kastner
Breaking Integrated Circuit Device Security through Test Mode Silicon Reverse Engineering
21st ACM Conference on Computer and Communications Security (ACM CCS), November 3-7, 2014, Scottsdale, Arizona, USA

Semi-invasive Probing Attacks

- We use tiny probe needles to intercept/modify signals on chip
- With semi-invasive attack we can only contact existing probe pads
- Can be useful to access unbonded test pads



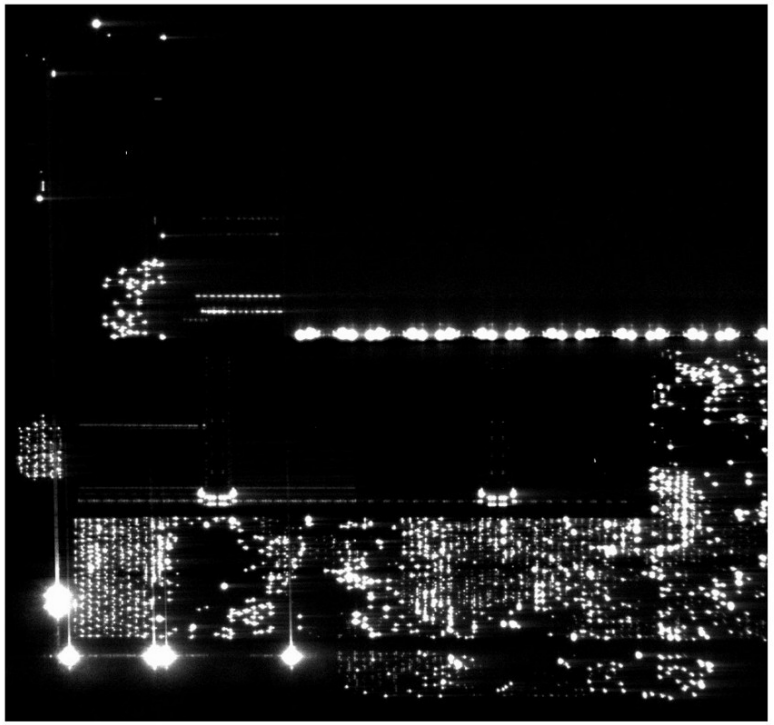
[wikipedia]

Backside Approach

- Silicon is transparent to infrared light
- We can easily open the IC from the backside (e.g., with Dremel tool)
- IR Laser probing and fault injection possible
  - Pro: metal layers do not obstruct laser beam, suitable for smaller designs
  - Con: NIR has longer wavelength (harder to focus), more expensive due to specialized IR optics

Photon Emission Analysis

- Backside attack
- Transistors emit IR light when they switch
- Bits (e.g. encryption key) are clearly visible
- Expensive IR micoscope necessary (e.g., Hamamatsu Phemos)
- Accumulation of many images

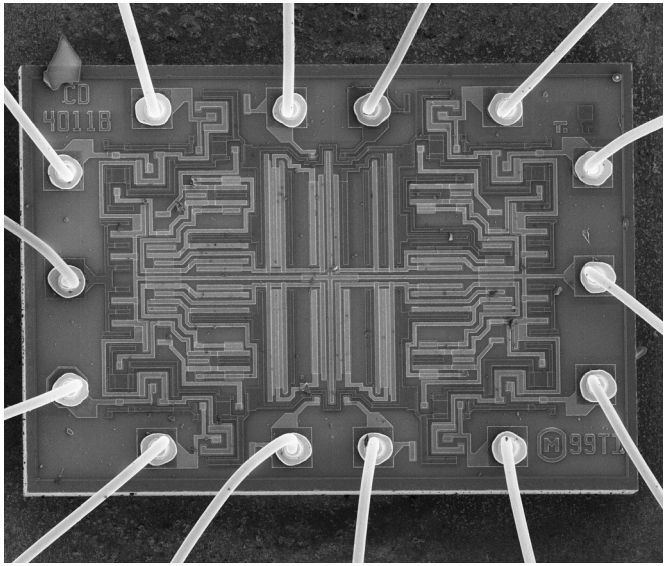


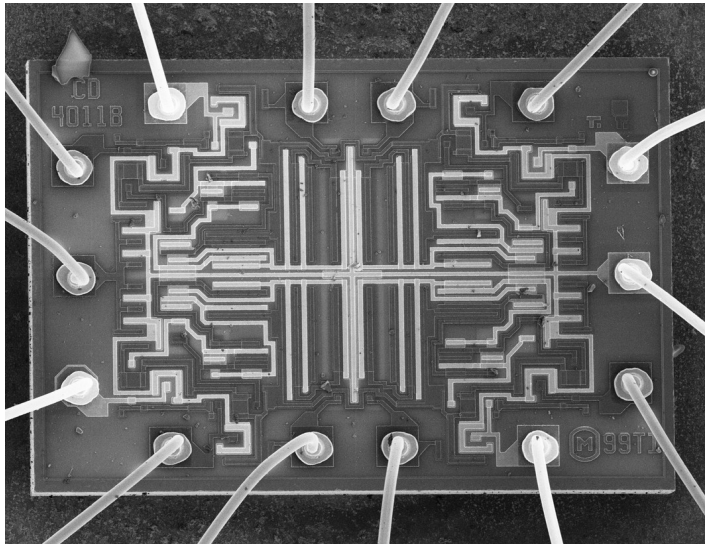
(b) Emission image of ATMega SRAM

Source: Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, Jean-Pierre Seifert
Simple Photonic Emission Analysis of AES, Photonic Side Channel Analysis for the Rest of Us
Cryptographic Hardware and Embedded Systems – CHES 2012

Voltage Contrast Imaging

- Idea: We supply a chip in the SEM with power (biasing)
- Electrons from beam will scatter differently if they hit biased signal on the die
- We can clearly see the difference between Vcc and GND





Localized Side Channel Attacks

- Idea: We use side-channel information such as photonic emissions or dynamic voltage contrast to obtain critical insight information
- We can focus in the areas of interest (e.g., crypto core, memories, bus)
- Examples:
  - Retrieve AES encryption key
  - Retrieve password transferred over a bus
  - Etc.

Semi-invasive Attack Conclusion

- We can get more insight than with non-invasive attacks
- Increased cost, but powerful attacks possible
- Limitation:
  - Only limited reverse engineering possible due to many layers of newer Ics
  - Knowledge of implementation internals might be necessary to mount attacks (newer chips – more design complexity)

Overview: Invasive Attacks

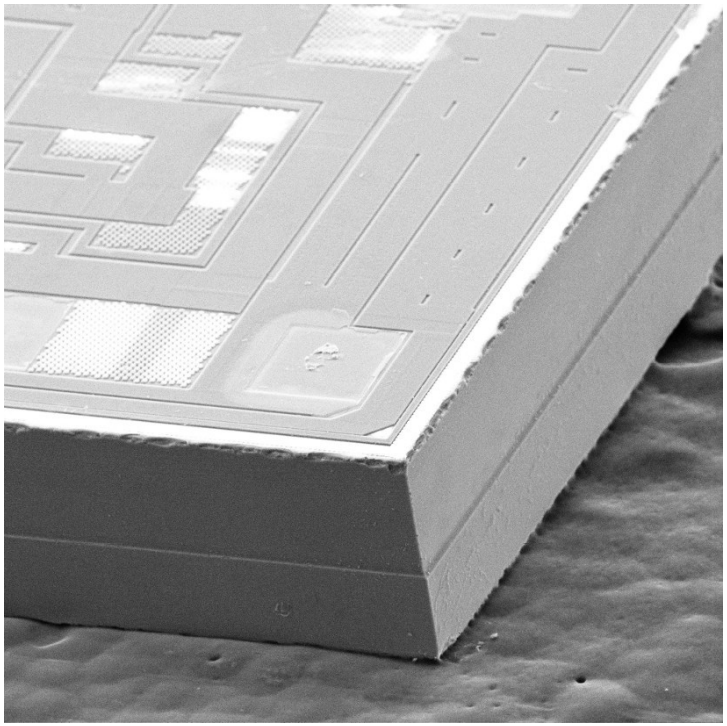
- Reverse Engineering and Deprocessing
- Probing Attacks
- Focused Ion Beam

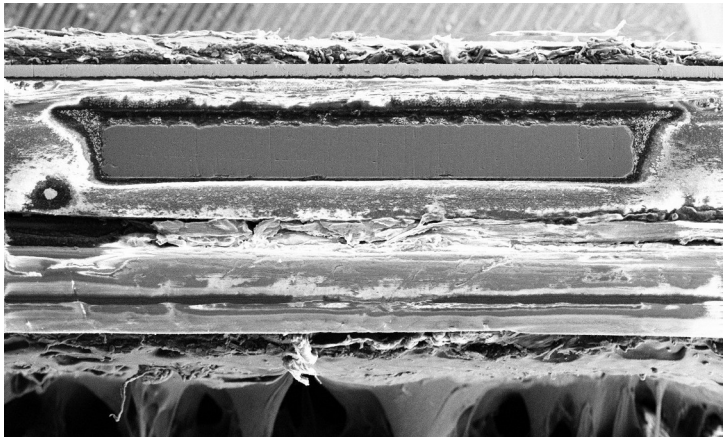
Invasive Reverse Engineering

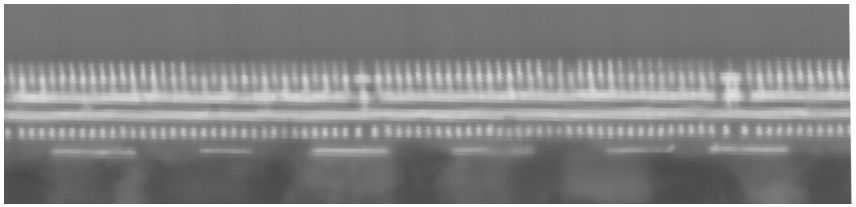
- We need to obtain deep internal know-how on the implementation internals
- Idea: We deprocess the die and analyze its layers
- *Combination* of different deprocessing techniques leads to success:
  - Wet chemical deprocessing
  - Plasma deprocessing
  - Polishing

Cross Sectioning

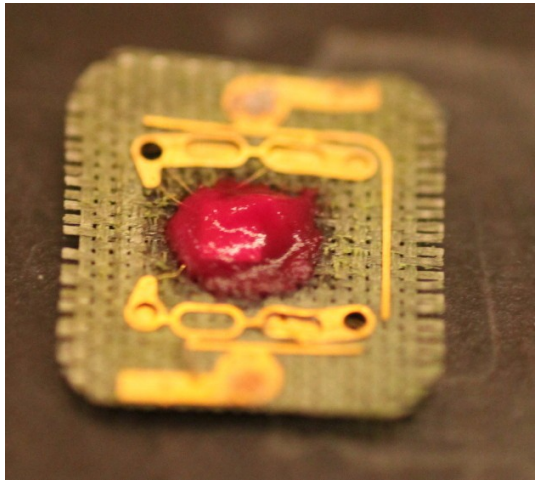
- We dissect a package/chip to identify its layers
- Helpful to obtain information for deprocessing







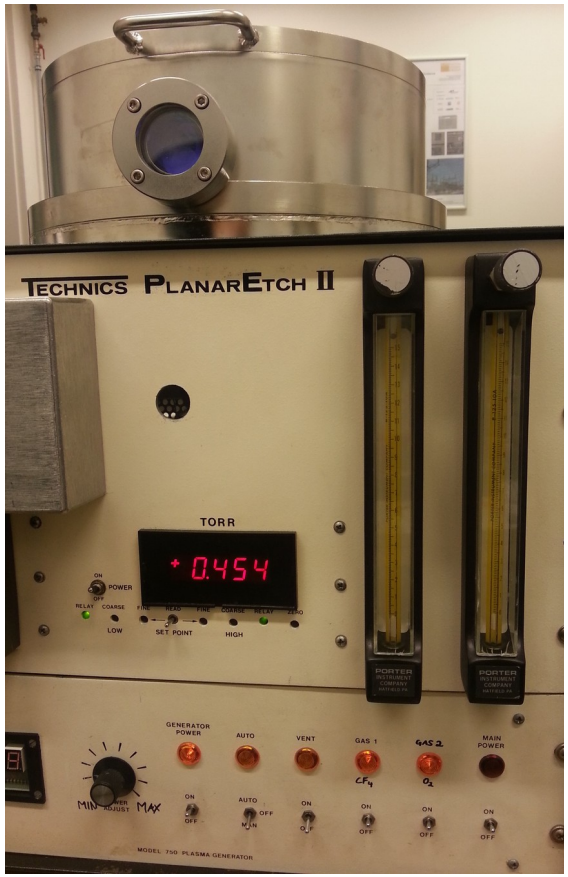
Wet Chemical Deprocessing



- Idea: Use material specific chemicals (e.g. acids) to selectively remove (parts of) an IC layer
- Problem: Isotropic etching, different etch speeds, unwanted corrosion of other materials, underetching
- Mitigation: Use of custom etching gels

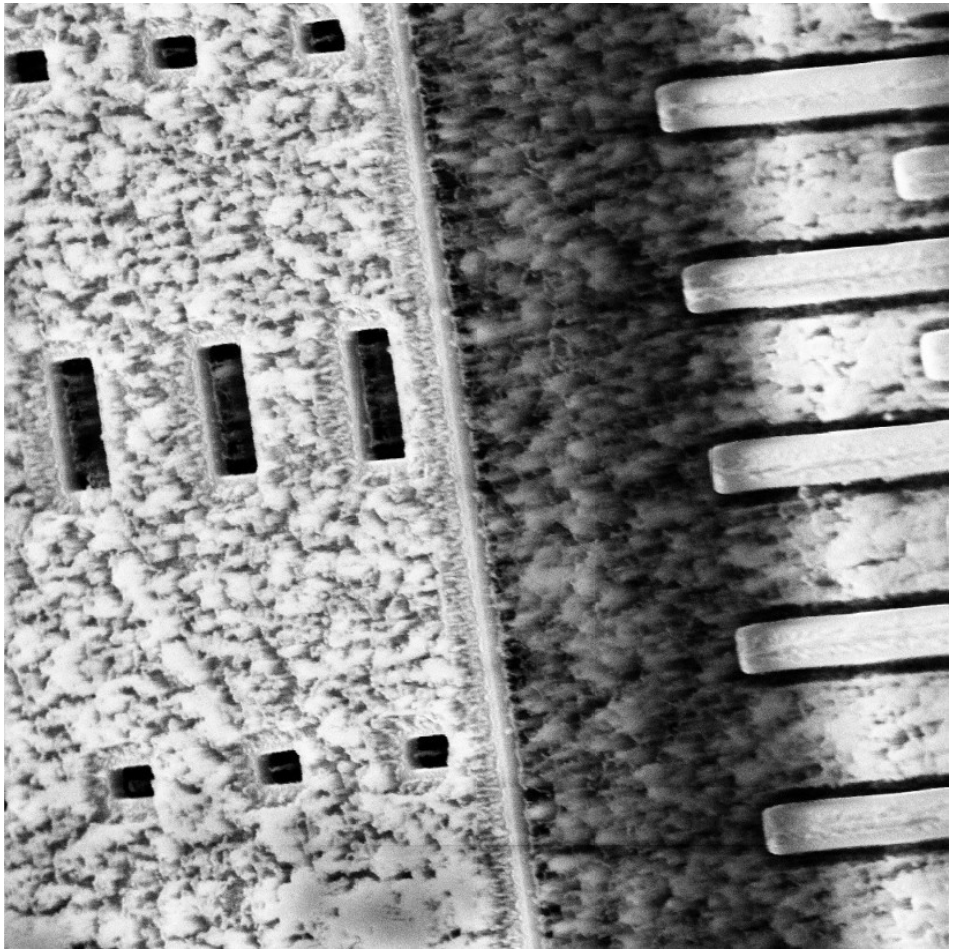
Plasma Deprocessing

- Principle of plasma etching already covered (i.e., plasma decapsulation)
- Advantage:
  - Very clean results
  - strong selectivity
  - passivation removal
- Disadvantages:
  - Metal etching requires highly toxic Chlorine based gases (we don't do that)
  - Formation of “RIE grass”



RIE Grass

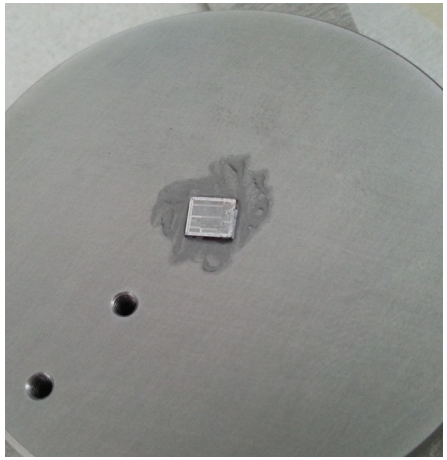
- Can be avoided by modifying plasma parameters (i.e., pressure, gas mixture, RF power)



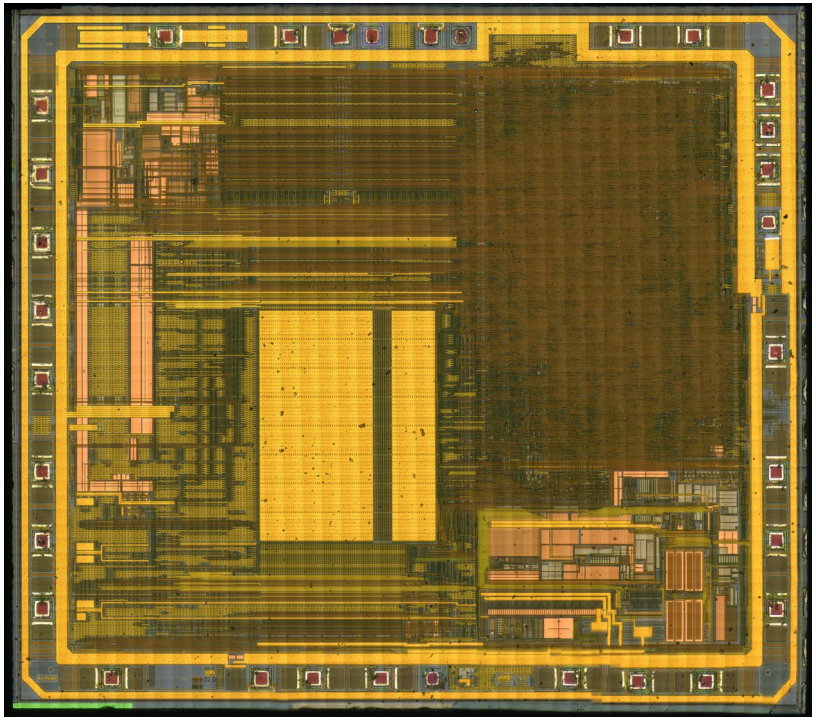
Polishing

- We use slurry with silica crystals for polishing
- Die is mounted with special wax
- Alignment is key to get planar polishing results

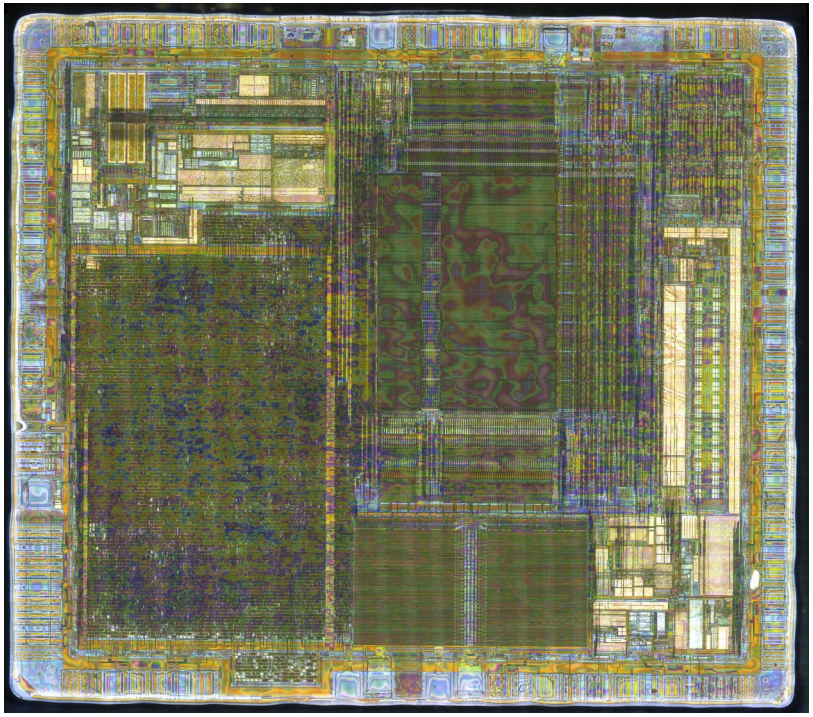
- Disadvantage:
  - Uneven results
  - Material dependent removal rates



Example: Metal Layer Removal

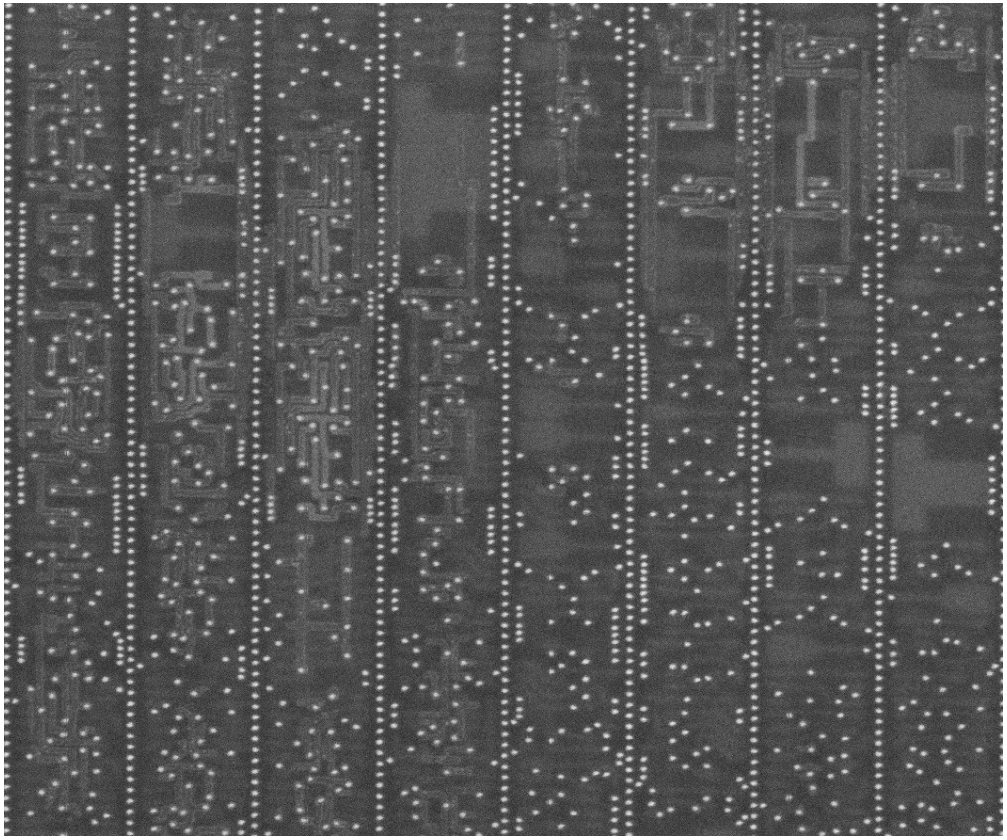


Top metal layer



Interconnect layer
exposed below

Example: Via Imaging



Result: Vias highlight well after correct die deprocessing

Dealing with Design Compexity

- Instead of manual image analysis, we can resort to semi-automatic computer vision approaches

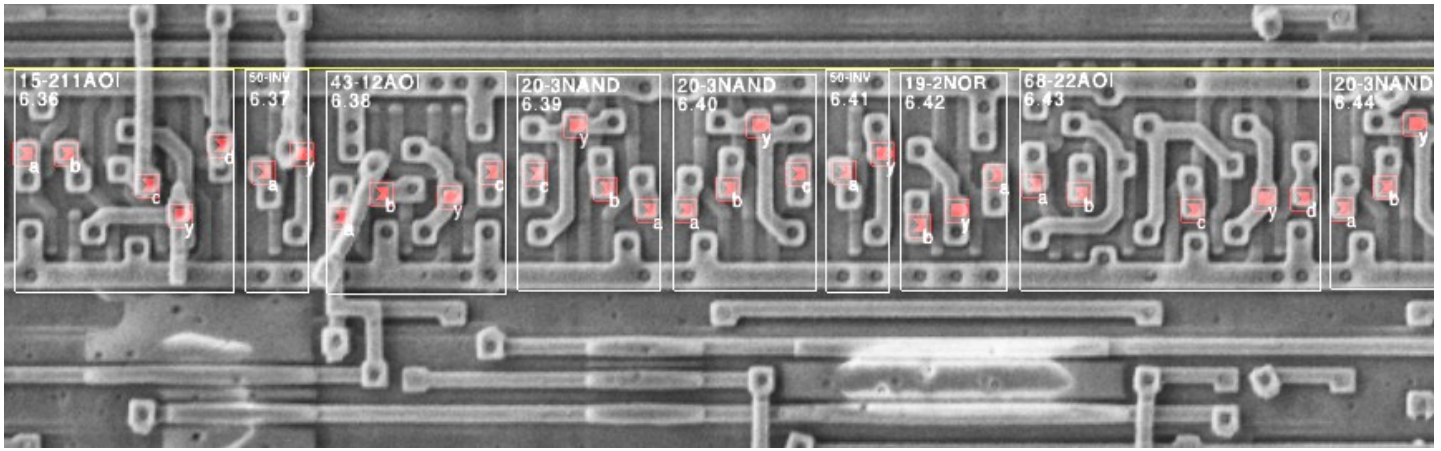
- Line tracing
- Via detection
- Gate recognition

Gate/Logic Recognition and Netlist Extraction

- Logic implemented in ICs usually consists of a number of “standard cells”
- Standard cells have low level logic functions (e.g., Inverter, XOR, AND, NAND, etc.)

Automated Gate Recognition

- Using pattern recognition, we can identify the standard cells and how they are interconnected
- We can reconstruct the implemented logic



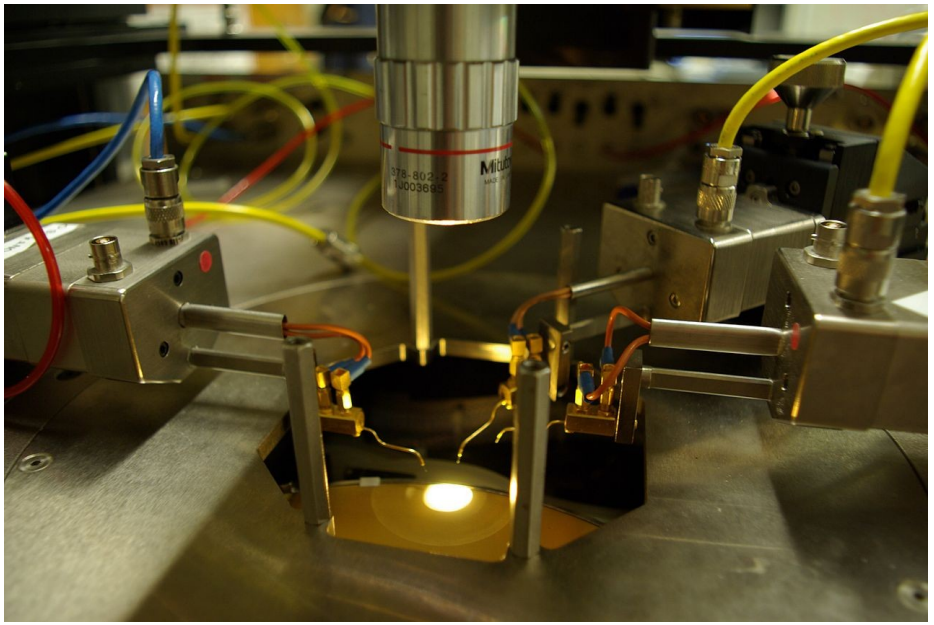
Source: degate.org

Invasive Probing Attacks

- Idea is the same as with semi-invasive probing
- Key difference: We can now not just probe existing pads, but arbitrary signals

- Usable for full code extraction

- Newer chips:
  FIB required



[wikipedia]

Focused Ion Beam (FIB)

- Similar to a Scanning Electron Microscope
- Uses Ga ions instead of electrons (more mass)
- Can be used for IC circuit edits:
  - Cut traces
  - Deposit new ones
  - Make probe pads
  - “Hotwire” security fuses
  - Also from backside through bulk Si
- There are 2 at TU-Wien, but only 1 can deposit SiO2
  - So far :-)

Probing Countermeasures

- Shields
- Sensor meshes around chip
- Sensors
- Special materials that make it hard to access the IC without damage

Silicon RE Countermeasures

- “Glue logic design”
  - Instead of highly structured layout, chip structure is randomized to make reverse engineering harder
  - Useless against pattern recognition
- Bus scrambling and encryption
  - Bus lines are mixed up (scrambling)
  - Possible use of encryption/decryption logic (e.g. XOR)

The End

Thank you for your attention!

Hope it wasn't too much HWSec :)