

A Whitebox Introduction to Fault Attacks

Patrick Schaumont

schaum@vt.edu

Professor



Karine Heydemann

karine.heydemann@lip6.fr

Associate Professor



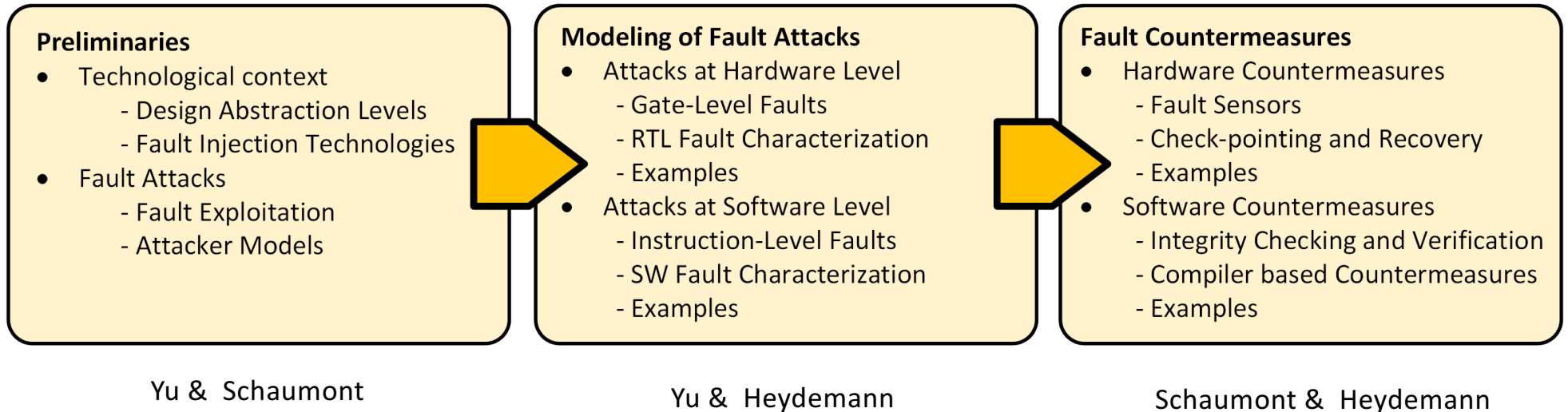
Qiaoyan Yu

qiaoyan.yu@unh.edu

Associate Professor



Outline of This Tutorial



Part 1: Preliminaries

---Technological Context

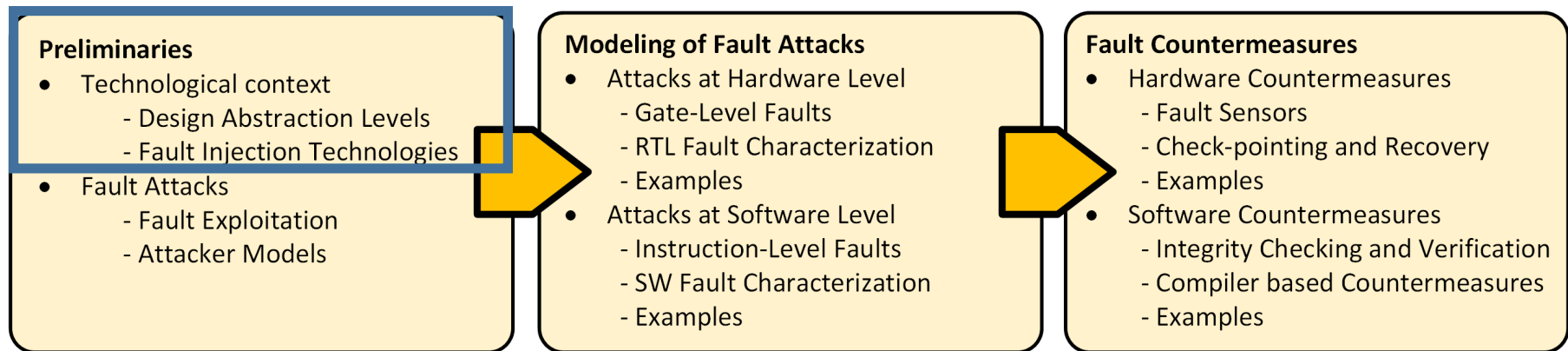
Qiaoyan Yu

qiaoyan.yu@unh.edu

Associate Professor



**University of
New Hampshire**



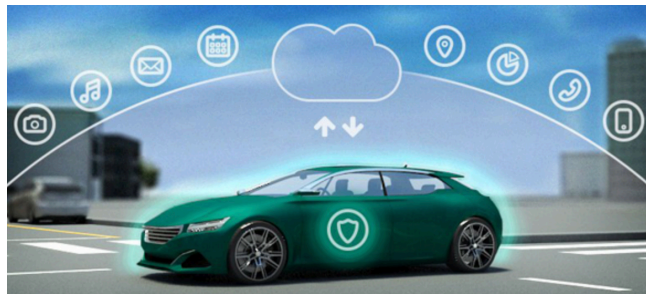
Outline

- Motivation examples
- Design abstraction levels
- Relevant technologies of interest for fault attacks
 - Timing violations
 - Noise injection
- Equipment for practical fault attacks
- Examples of fault attack experiments

Security Concerns



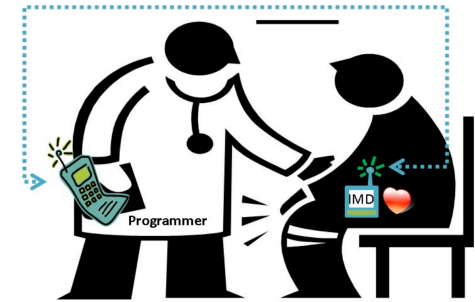
Credit card



Connected vehicle



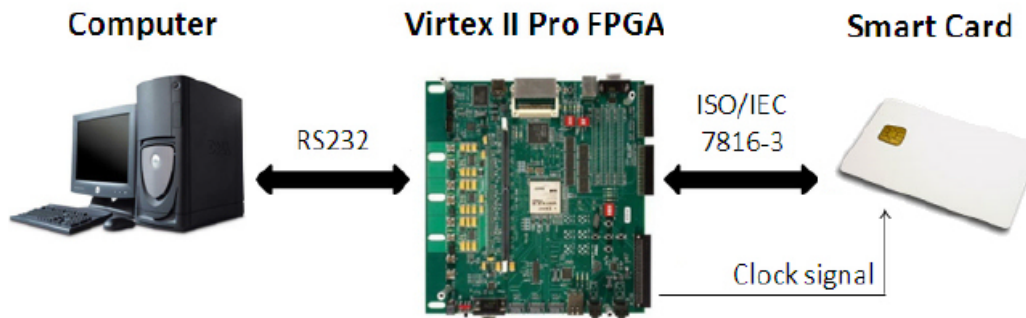
Smart home



Implanted device



A Practical Example



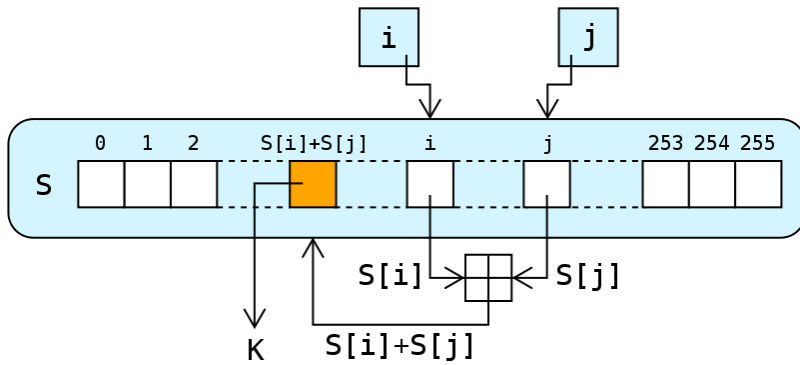
- A target platform a MCU - ATmega163 microcontroller
- FPGA is used to generate the clock for MCU
- Due to the glitch, the instruction EOR R15, R5 never executed
 - MCU does not have enough time to load new command from program memory

Glitch period	Cycle	Instruction	Opcode (bin)
-	i	NOP	0000 0000 0000 0000
-	$i+1$	EOR R15, R5	0010 0100 1111 0101
≤ 59 ns	$i+1$	NOP	0000 0000 0000 0000

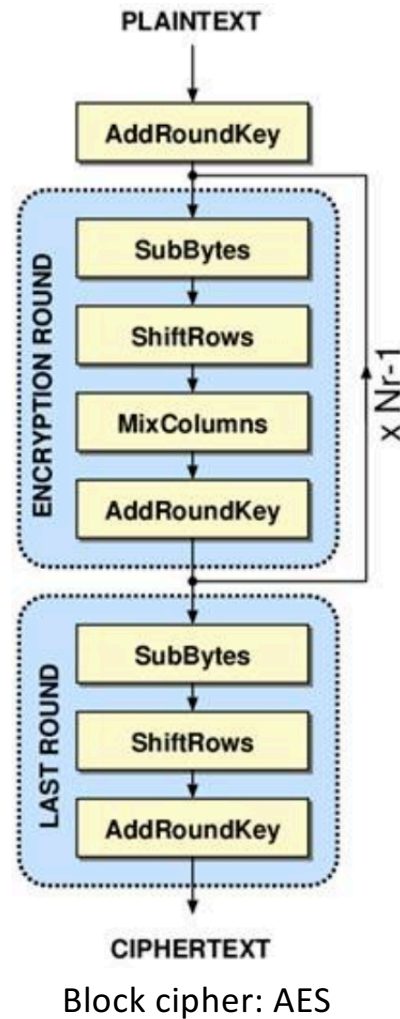
Glitch injection in Clock

Various Ciphers

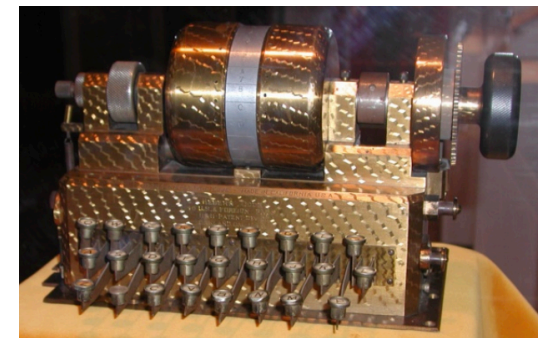
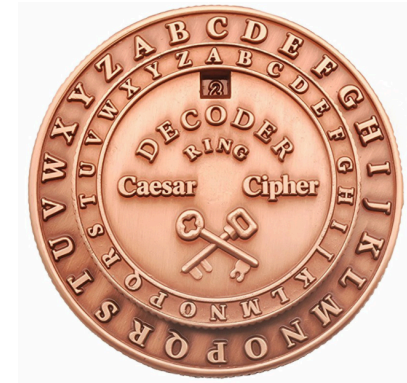
- Ancient cryptography
- Block cipher
- Stream cipher



Stream cipher: RC4



Block cipher: AES

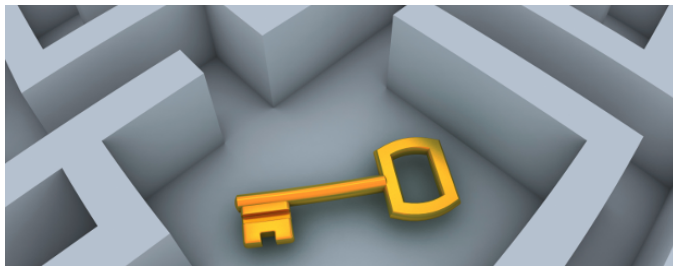


A single-rotor Hebern machine

Cracking Cryptographic Key

[1]

Key Size	Possible Combinations
56-bit (DES)	7.2×10^{16}
128-bit (AES)	3.4×10^{38}
192-bit (AES)	6.2×10^{57}
256-bit (AES)	1.1×10^{77}



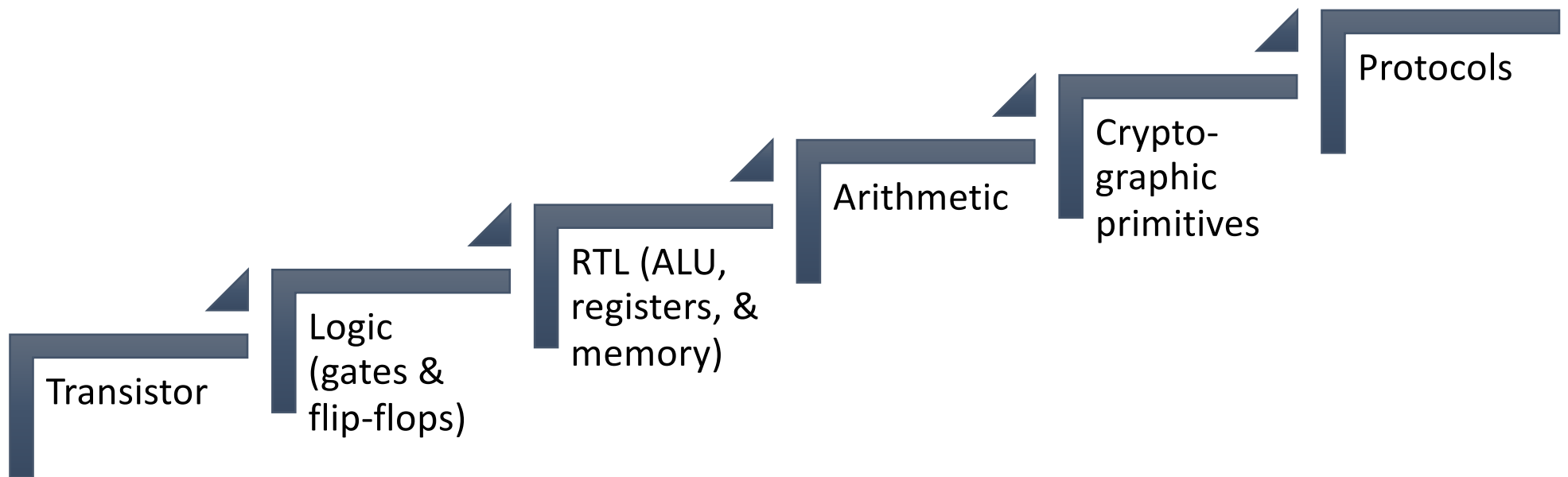
7.66×10^{25} Years [2]



[1] M. Arora, online, July 2012.

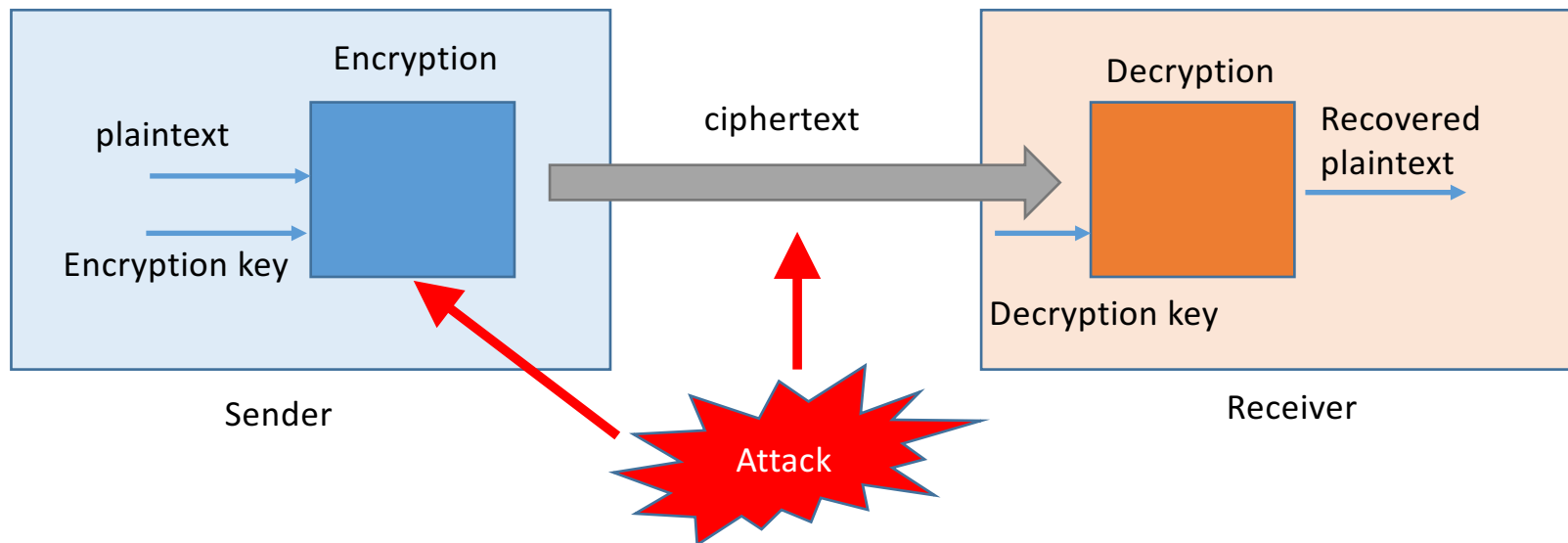
[2] Seagate, Technology paper, 2008.

Abstract Levels in Cryptographic Hardware Design Process

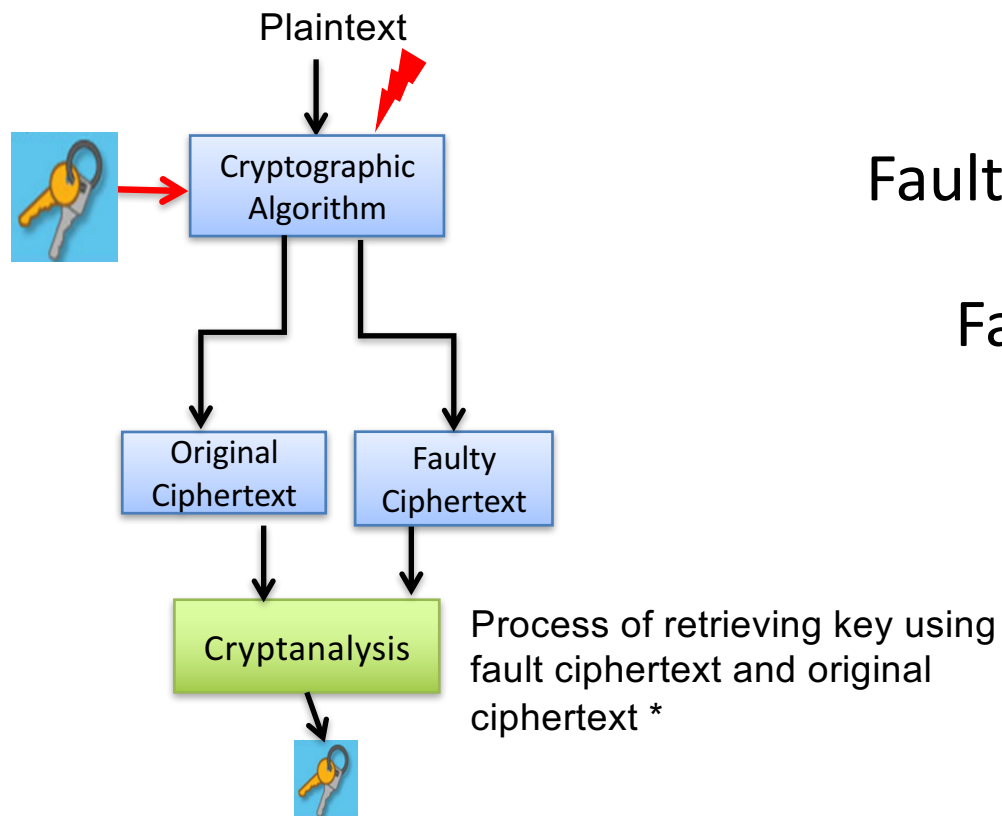


Fault Attack on Cryptosystems

- Goal: retrieve the crypto key
- Active attacks against cryptographic implementations
 - A fault can cause errors → An errors can be exploited to expose secrets



Fault Attack for Key Extraction



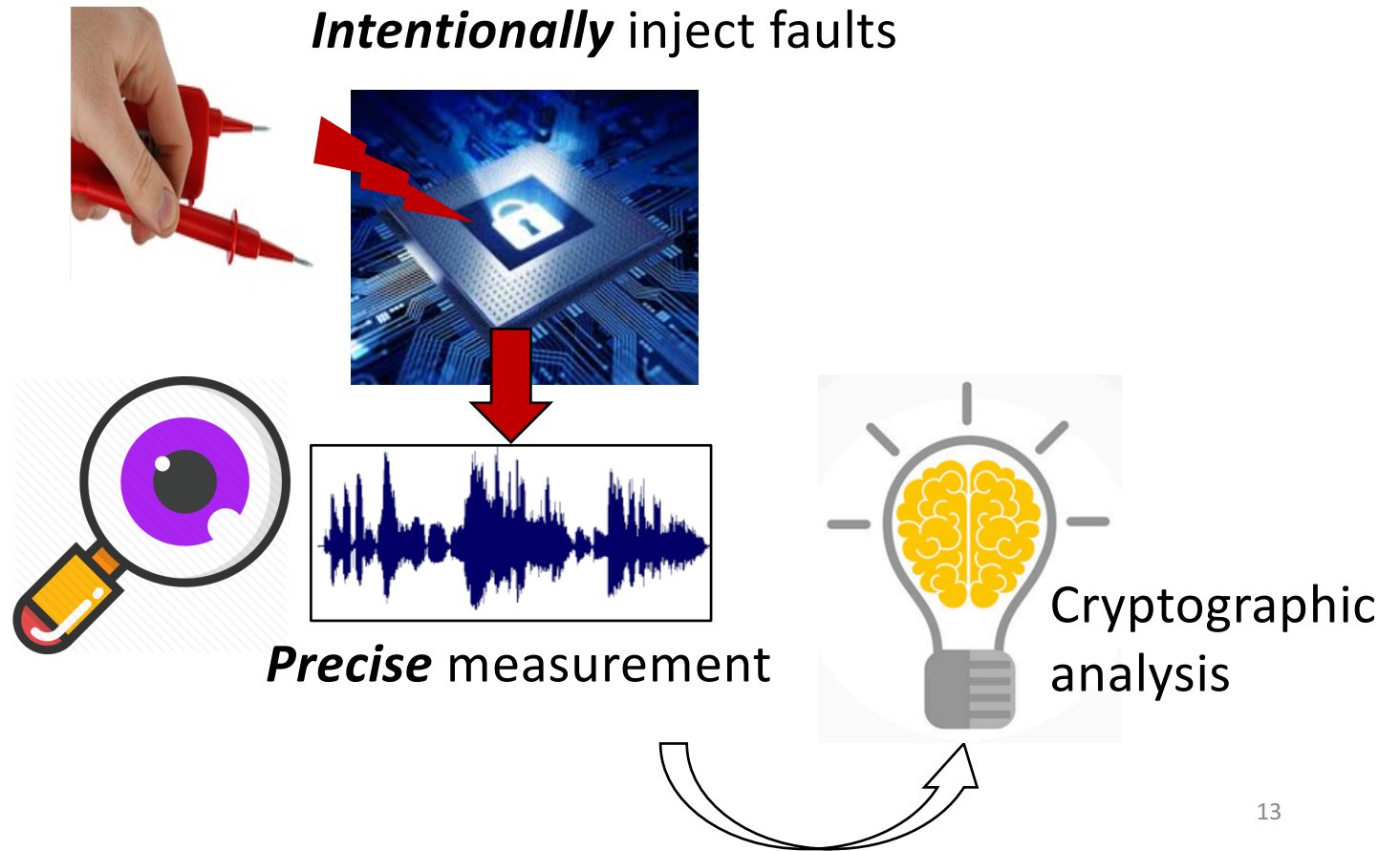
Fault-free version: $a \text{ xor } \text{key} = Y$

Faulty version: $a' \text{ xor } \text{key} = Y'$

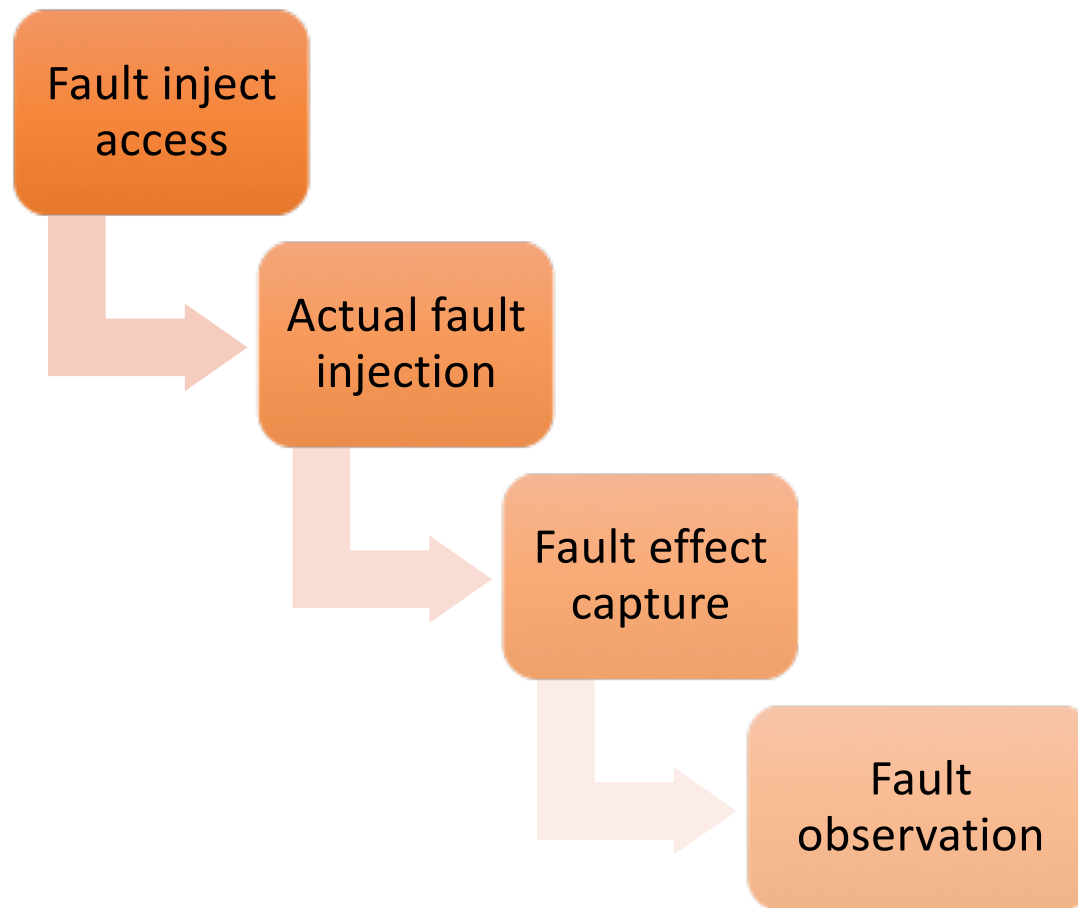


$$\text{key} = a \text{ xor } Y$$

Fault Attack Flow



Physical Implementation Procedure for Fault Attacks



Non-intentional Fault vs. Intentional Fault

- Random faults are non-intentional faults
 - External source
 - Internal source
- Faults induced by fault attack are intentional faults

Impact scope

- Global attack
- Local attack

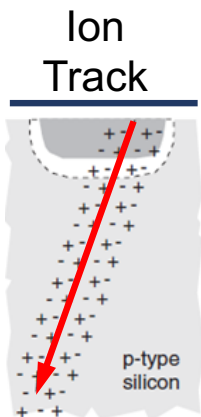
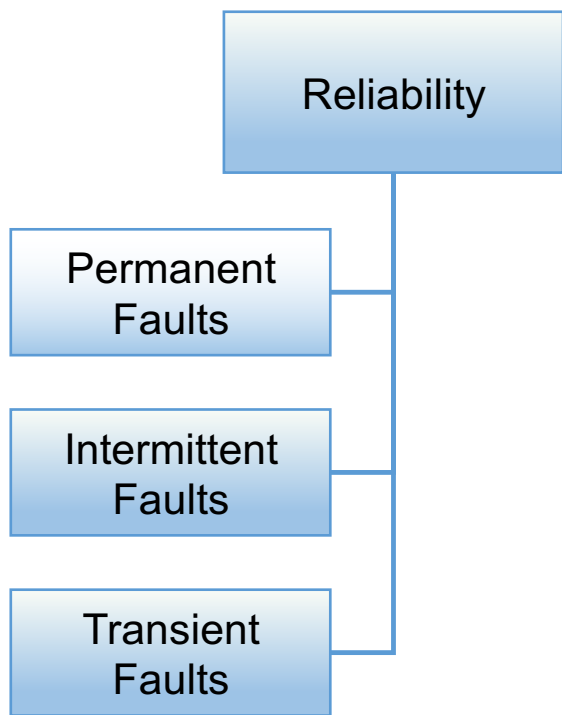
Fault effect

- Timing violations
- Noise injection

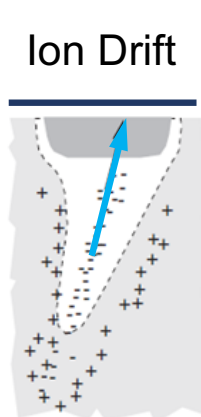
Attack means

- Non-invasive attack
- Semi-invasive attack
- Fully Invasive attack

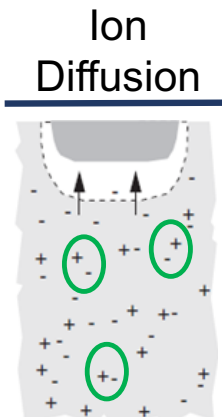
Non-Intentional Faults on Integrated Circuits



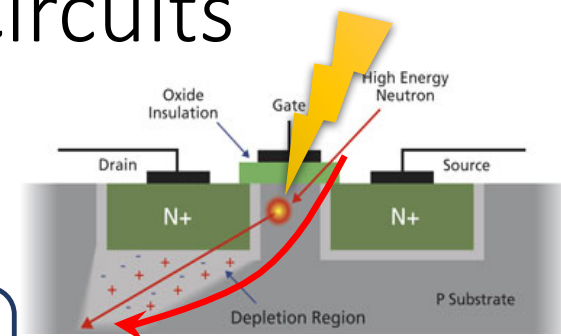
Ionizing the path and creating electron-holes



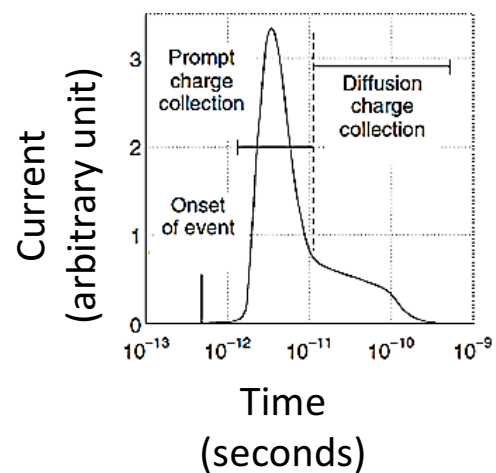
Electric field collects the carriers and creates a current glitch



The electrons diffuse in to the depletion region and recombine

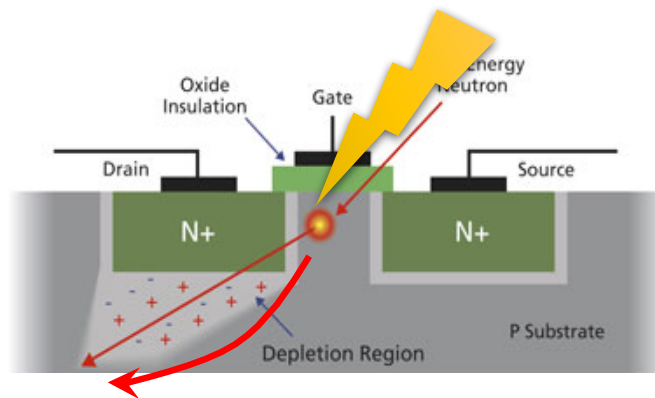
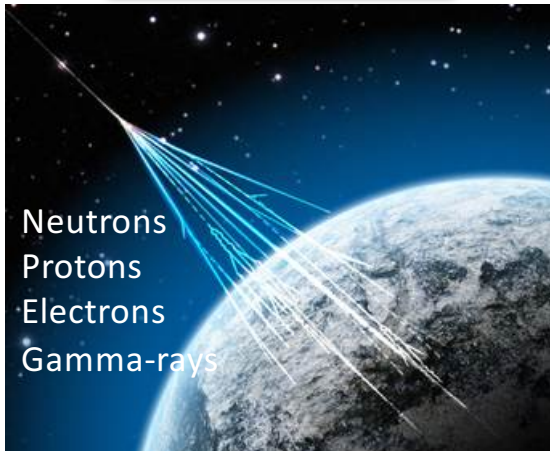


Corresponding Current Pulse



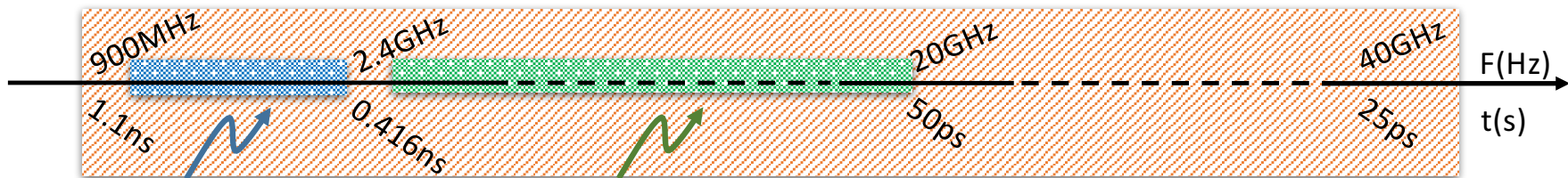
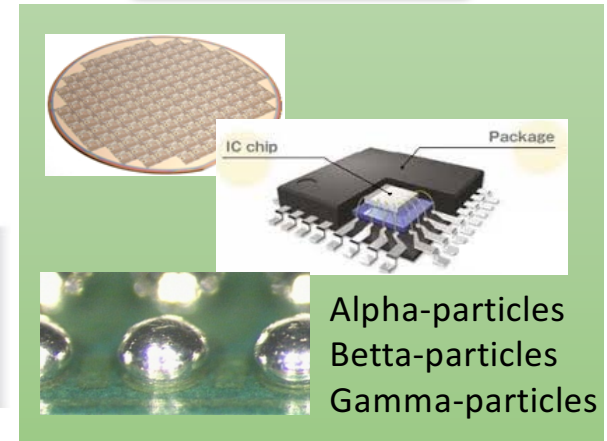
Source of Transient Faults

[1] External Source



Internal Source

[2]



- Cellular telephones
- Pagers
- Radar systems
- Satellite communications

25ps(65nm) < SE Pulse width < 1ns(130nm) [3,4]

[1] <https://www.hep.ucl.ac.uk/creamtea/>

[2] S. Kumar, FDTC'12.

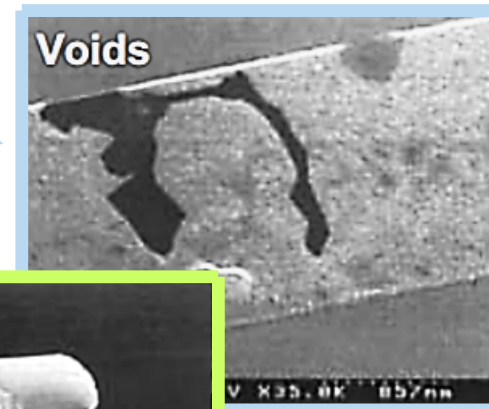
[3] M. J. Gadlage, TNS'11.

[4] B. Narasimham, TNS'07.

Source of Permanent Faults

Depletion of atoms (Voids):

- Slow reduction of connectivity
- Interconnect failure



Deposition of atoms (Hillocks, Whisker):

- Short cuts



Non-intentional Fault vs. Intentional Fault

- Random faults are non-intentional faults
 - External source
 - Internal source
- Faults induced by fault attack are intentional faults

Impact scope

- Global attack
- Local attack

Fault effect

- Timing violations
- Noise injection

Attack means

- Non-invasive attack
- Semi-invasive attack
- Fully Invasive attack

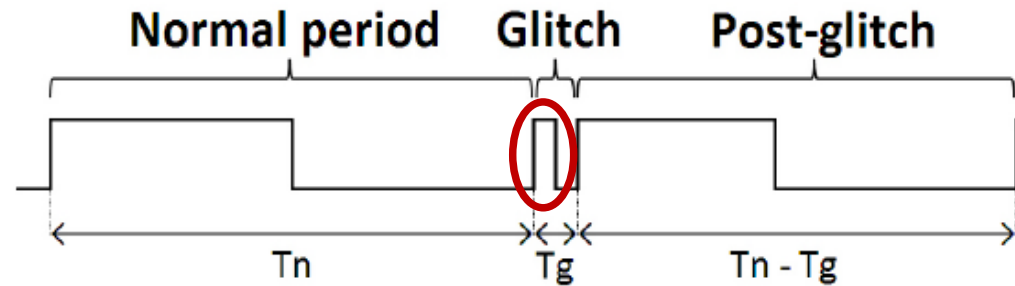
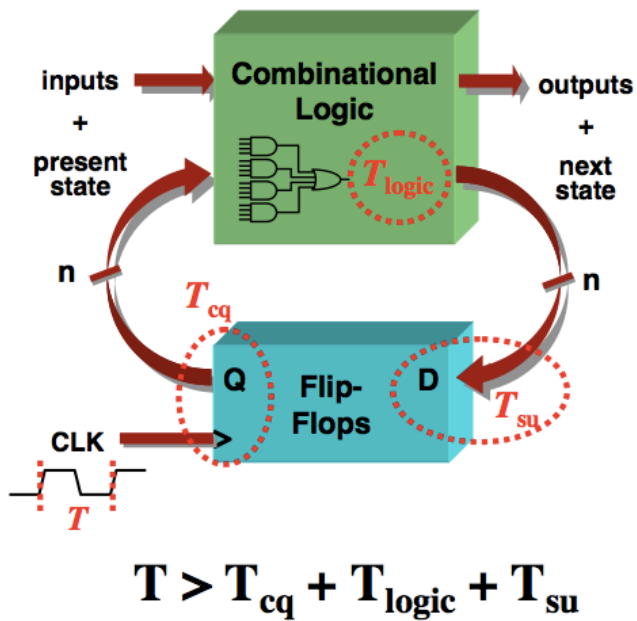
Global Attack vs. Local Attack

- Disturb all the entire netlist/chip simultaneously
 - Attack methods
 - Over-clocking
 - Under-powering
 - Heating
- Target a specific zone of the components' surface, rear or front
 - Attack methods
 - A laser beam
 - A particle source
 - Strong eddy currents

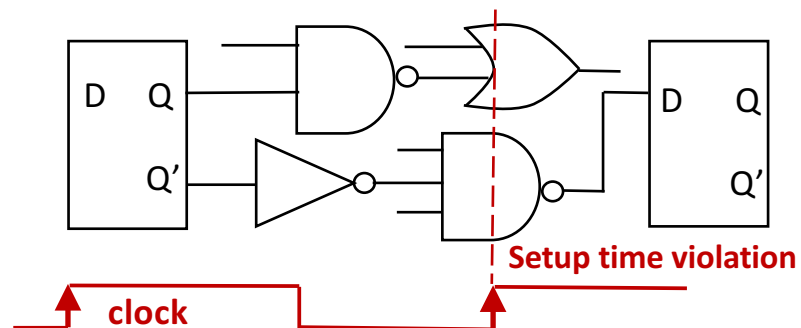
Feature	Global attack	Local attack
<i>Cost in equipment</i>	Low	High
<i>Required expertise</i>	low	High
<i>Easiness of detectability</i>	Yes	No
<i>Controllability in space</i>	No	Yes
<i>Controllability in time</i>	Yes	Yes

Fault Effect: Timing Violation

- Timing violation includes setup time or hold time violation



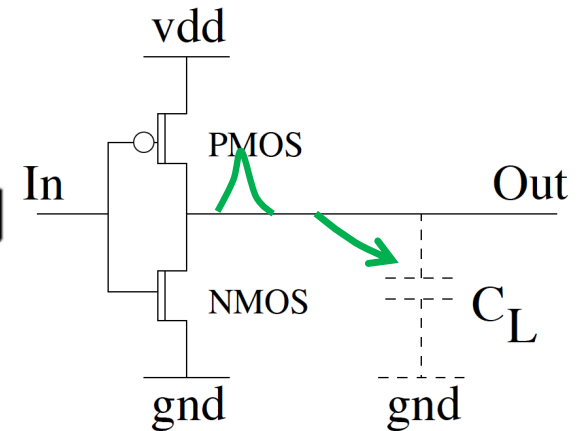
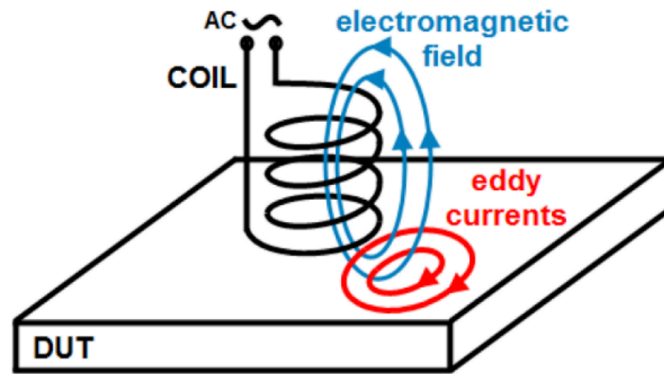
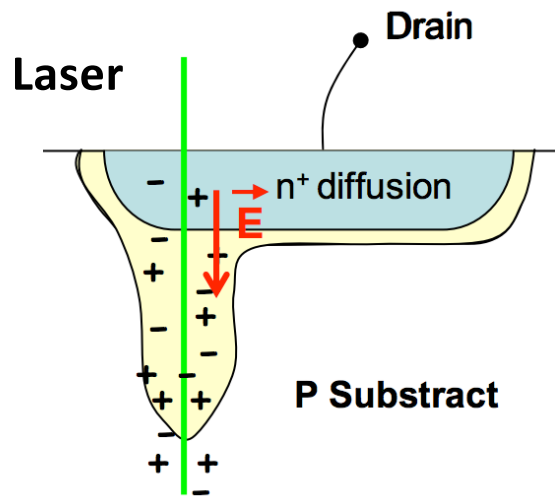
Glitch injected in clock



Setup time violation

Fault Effect: Noise Injection

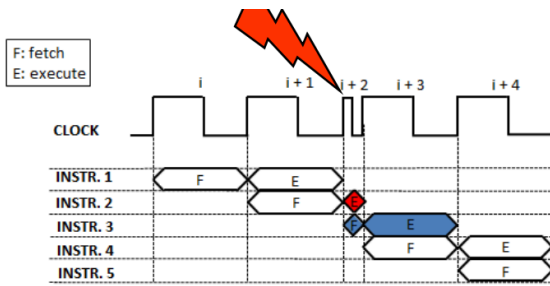
- A laser beam and an electromagnetic field can generate a transient pulse, forming a transient fault



Non-invasive Attack

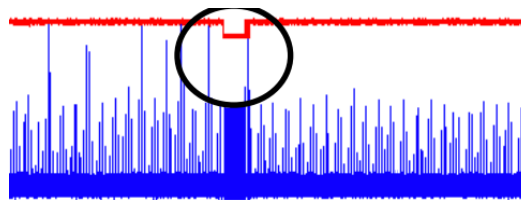
- Do not have physical damage to device
- Modify working conditions
- Moderate knowledge/equipment

Clock glitches



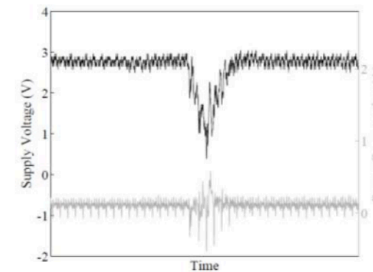
J. Balasch et al., FDT, 2011.

Under-powering



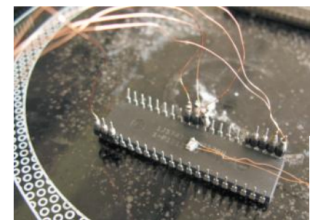
J. Balasch, CT-RSA'12.

Voltage spikes



M. Hutter, CHES'12.

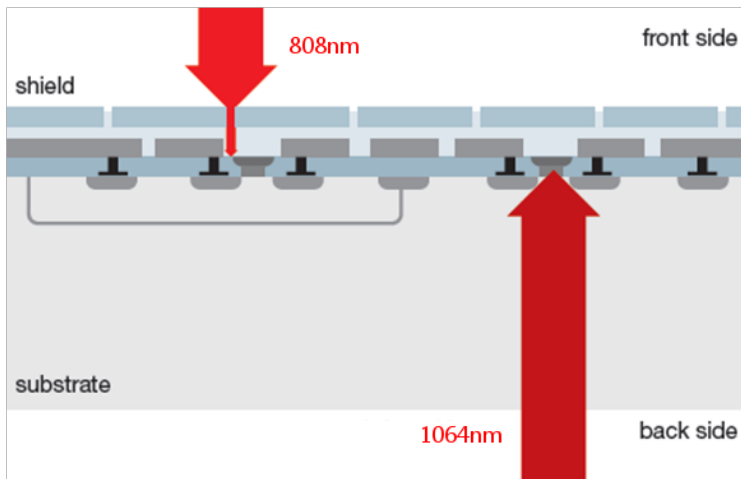
Temperature



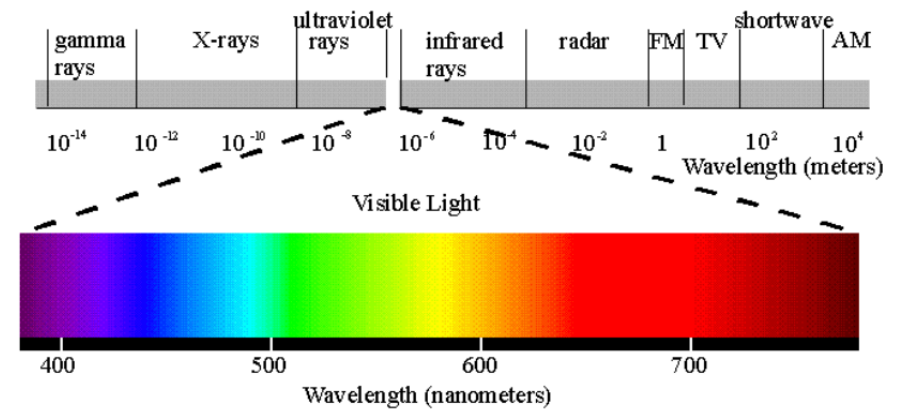
M. Hutter, CARDIS'13.

Semi-invasive Attack

- Chip de-capsulation
- Milling, etching, cleaning
- Affordable equipment

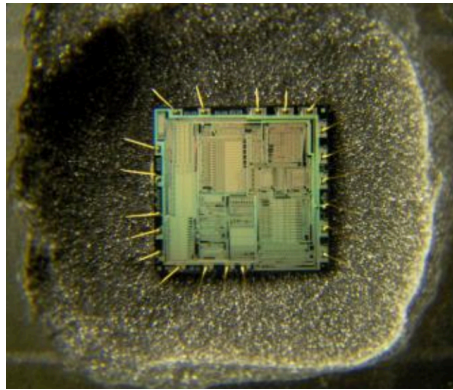


Woudenberg, FDTC'12.

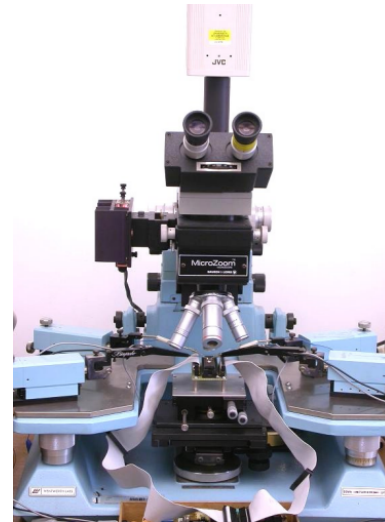


Fully Invasive Attacks

- Delayer or modify the chip
- Microprobe and internal fault injection
- Expensive equipment



http://en.wikipedia.org/wiki/File:Yamaha_YM3812_audio_IC_decapsulated.jpg



S. Skorobogatov, HWA'11.

Fault attack
model

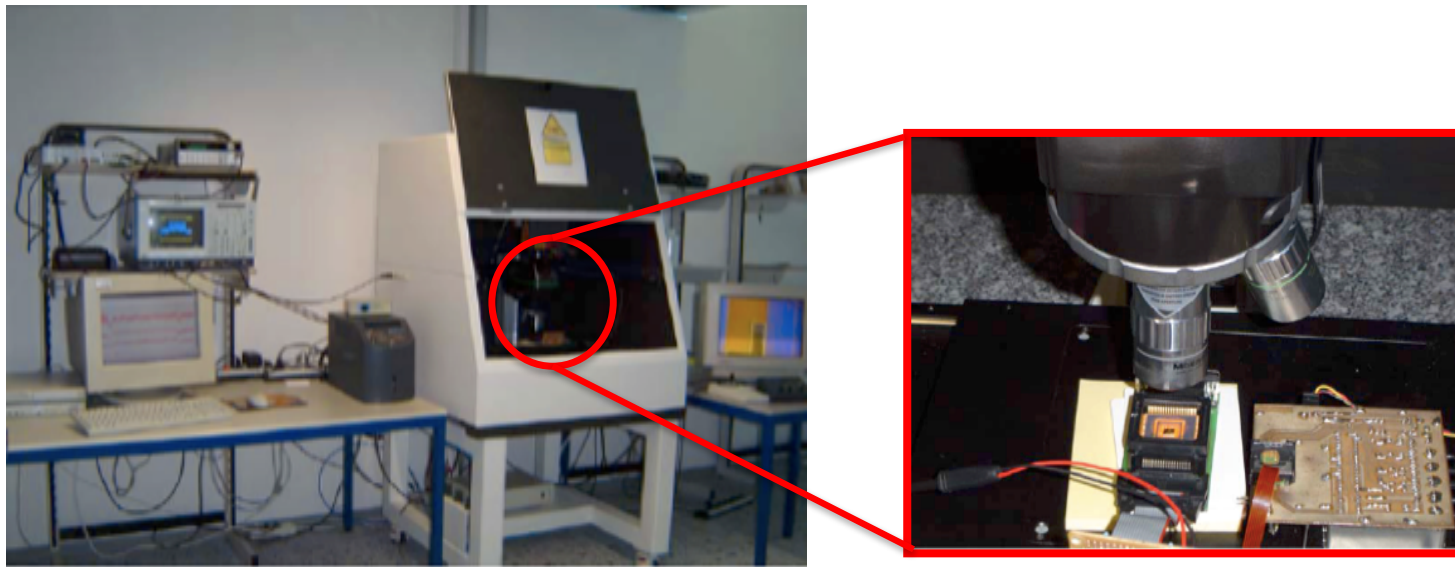


Physical
implementation
of fault injection

Feasibility of Fault Attacks

Fault Attack	Requirements on Fault Injection				
	Fault Granularity	Fault Type	Fault Injection Space	Fault Injection Timing	Fault Duration
Diff. fault attack	Bit, word	Random	Loose control	Loose control	Transient
Attacks on program flow	Bit, word, variable	Bit flip, set value	Strong control, Loose control	Strong control	Transient, Permanent
Algorithm specific attack	Bit, word, variable	Bit flip, set value	Strong control, Loose control	Strong control, Loose control	Transient, Permanent
Safe-error attacks	Bit, word, variable	Random	Strong control, Loose control	Strong control	Transient

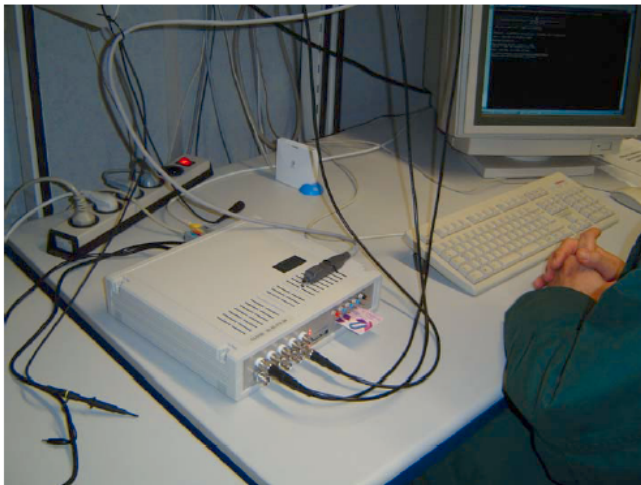
Laser Injection Station



Thanks to Dr. David Naccache, Vice President,
Gemplus Card International

CLIO Glitch Injector

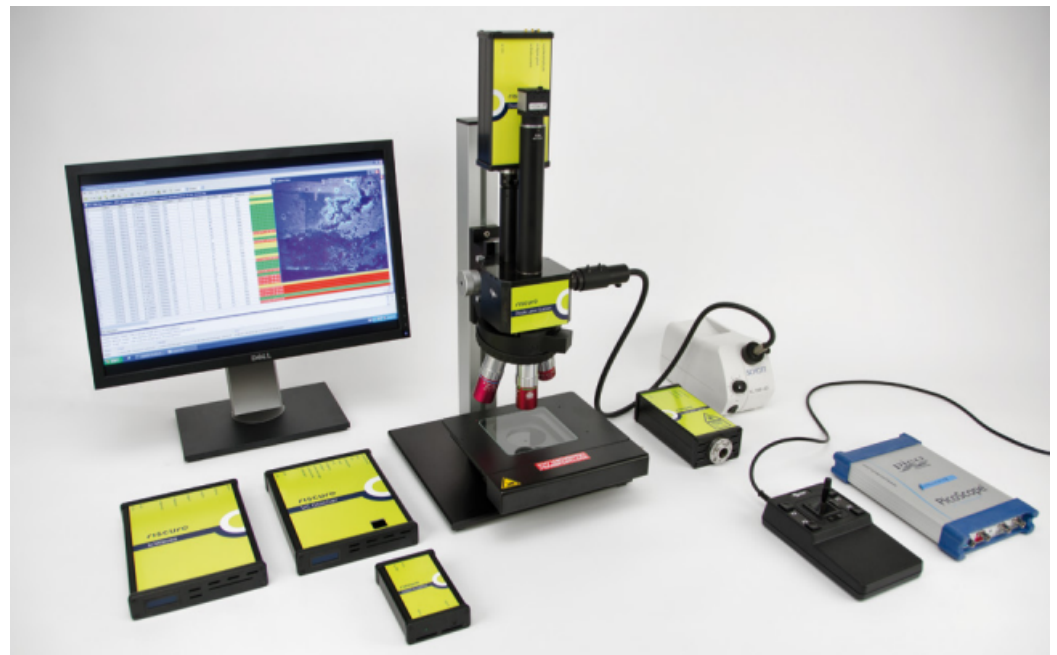
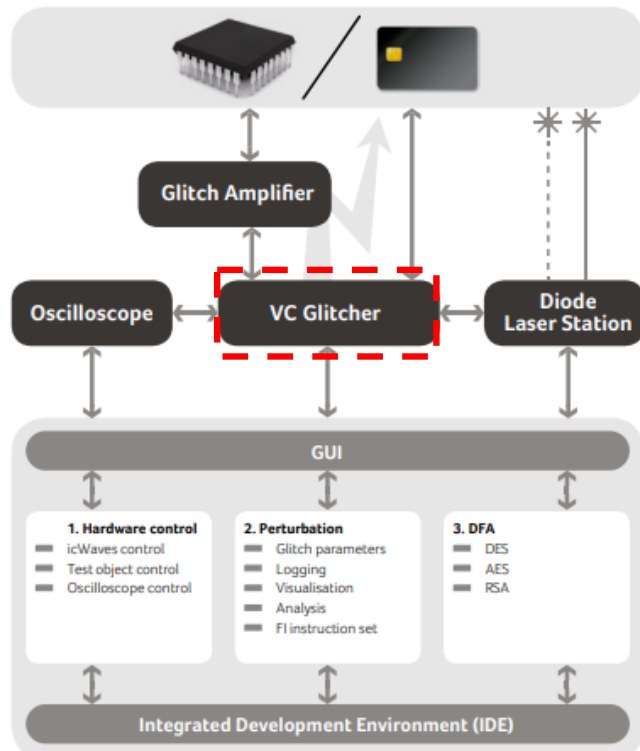
- Induce errors in the device
- Perform encryption with and without presence of fault
- Recover sensitive information



D. Naccache, IACR'04.



VC Glitcher



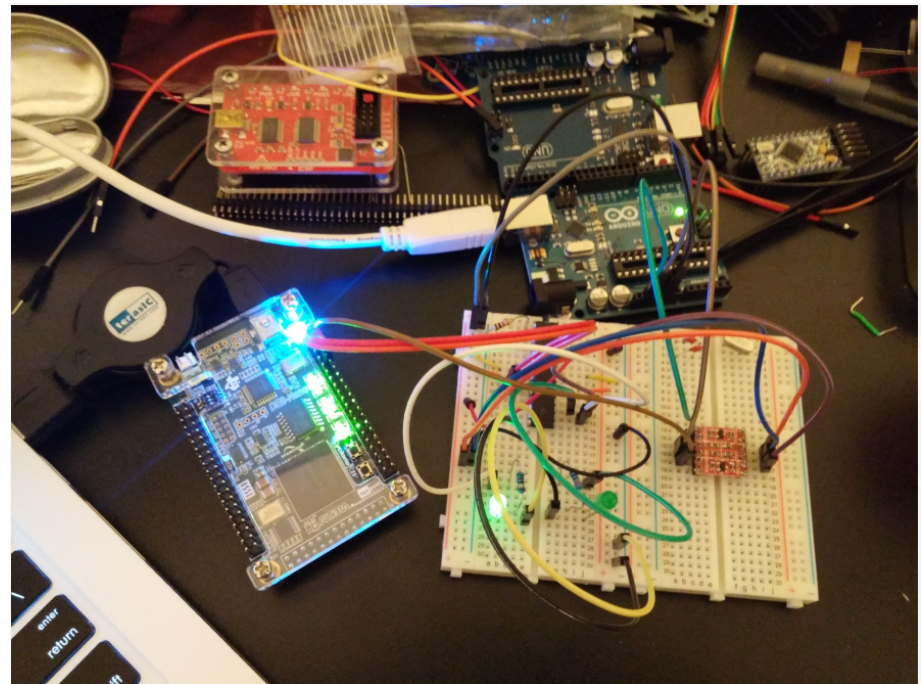
Inspector FI with VC Glitcher, Glitch Amplifier and Diode Laser Station

Experimental Setups for Clock Glitches

- DE0 Nano FPGA
- Target device - ATmega328p
- If glitch works then INFINITE_LOOP1 and INFINITE_LOOP2 should exit and run the normal program

```
START:
BSF  LED1          ; LED1 on and LED2 off
BCF  LED2
INFINITE_LOOP1:
GOTO INFINITE_LOOP1 ; Infinite loop. Exits only if glitching
                    ; works
BCF  LED1          ; LED1 off and LED2 on
BSF  LED2
INFINITE_LOOP2:
GOTO INFINITE_LOOP2 ; Infinite loop. Exits only if glitching
                    ; works
GOTO START
```

Demo example to insert the clock glitch



Laser Station 2

- Lab station 2 can perform advanced laser fault attacks
- It contains powerful Red and NIR diode lasers
- The special set of lasers with dedicated optics helps in precise fault injection
- The powerful laser can penetrate through the gaps in the shielding commonly used in today's secure chips

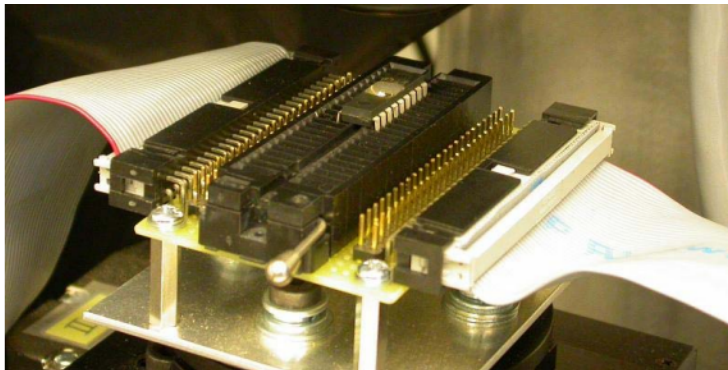
www.riscure.com



Laser Station 2

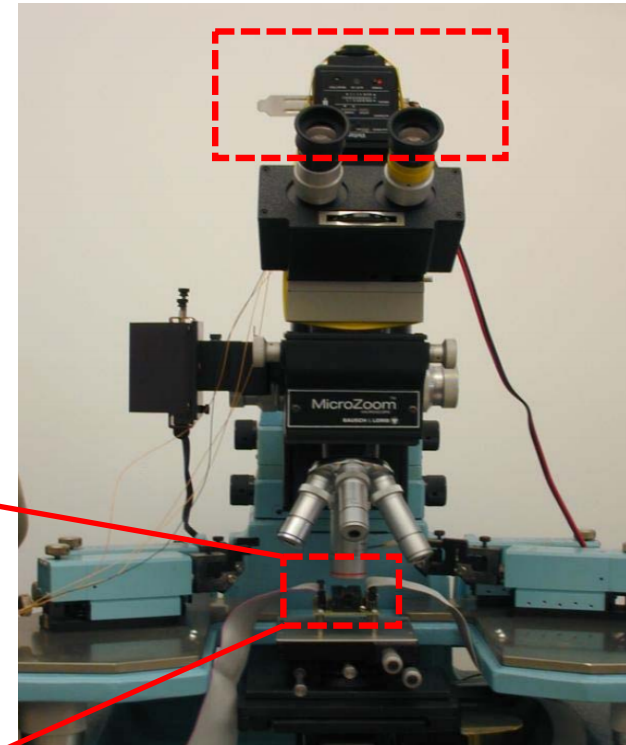
Experimental Setup for Optical Attacks

- The setup contains
 - Wentworth Labs MP-901 manual prober
 - Photoflash lamp (a Vivitar 550FD)
 - The test setup with the microcontroller
 - PIC16F84 in a test socket [2]



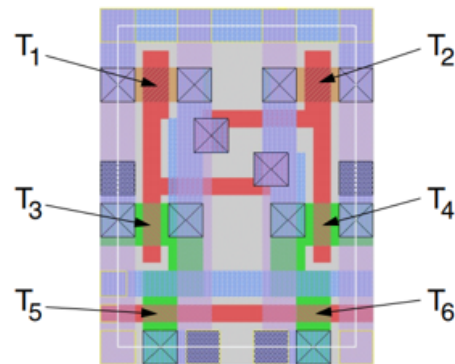
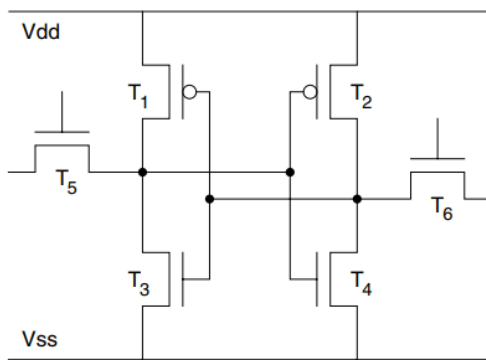
[1] S. Skorobogatov, CHES'02.

[2] S. Skorobogatov, FDTC'10.



Wentworth Labs MP-901 manual prober for optical attacks ^[1]

Example of an Optical Attack



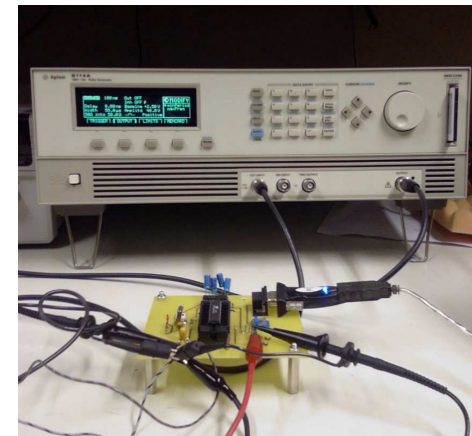
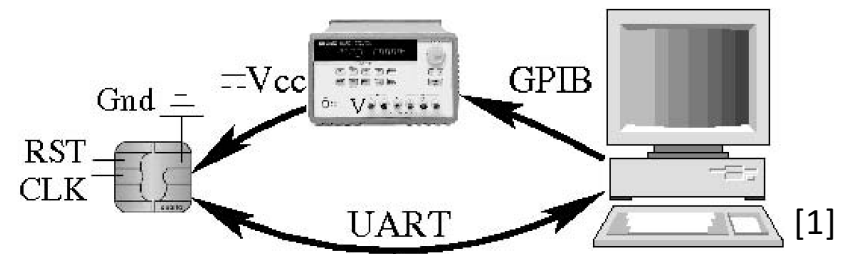
- If the transistor T3 and T4 could be opened for a very short time by an external stimulus (in this case optical source - flash), then it could cause the flip-flop to change the state

Circuit structure and layout of a six-transistor SRAM cell



Experimental Setups for Under-powering or Power Spikes

- Remotely controlled power supply providing successively values of V_{cc}
- Smart card with protection of AES
- AES fails when V_{cc} is lower than 800mV
- The tool generates glitches
- The chip is connected to computer

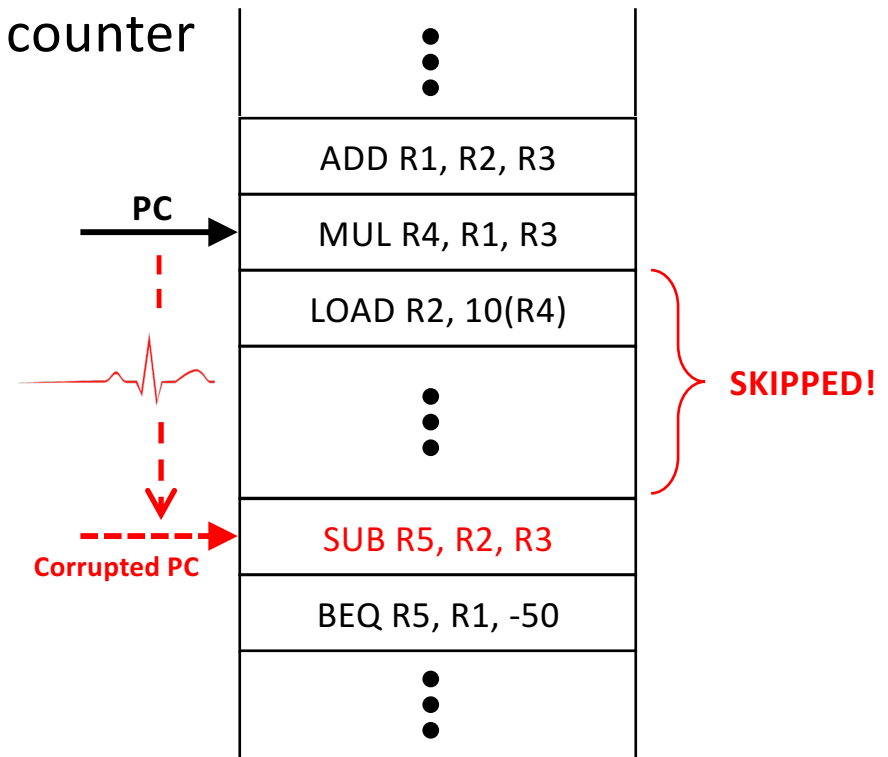


[1] N. Selmane, EDCC'08.

[2] Kim C.H., WISTP'07.

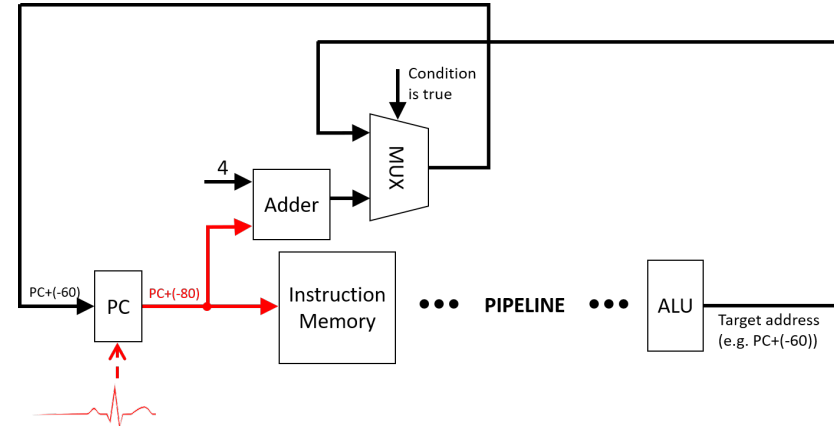
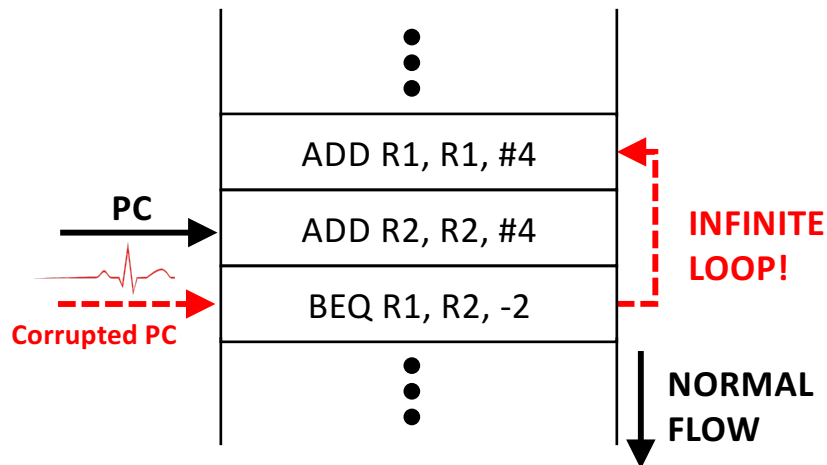
Program Counter Tampered by a Power Spike

- Power spike tampers with the program counter
- Pointing to unexpected instruction
- Original function being corrupted



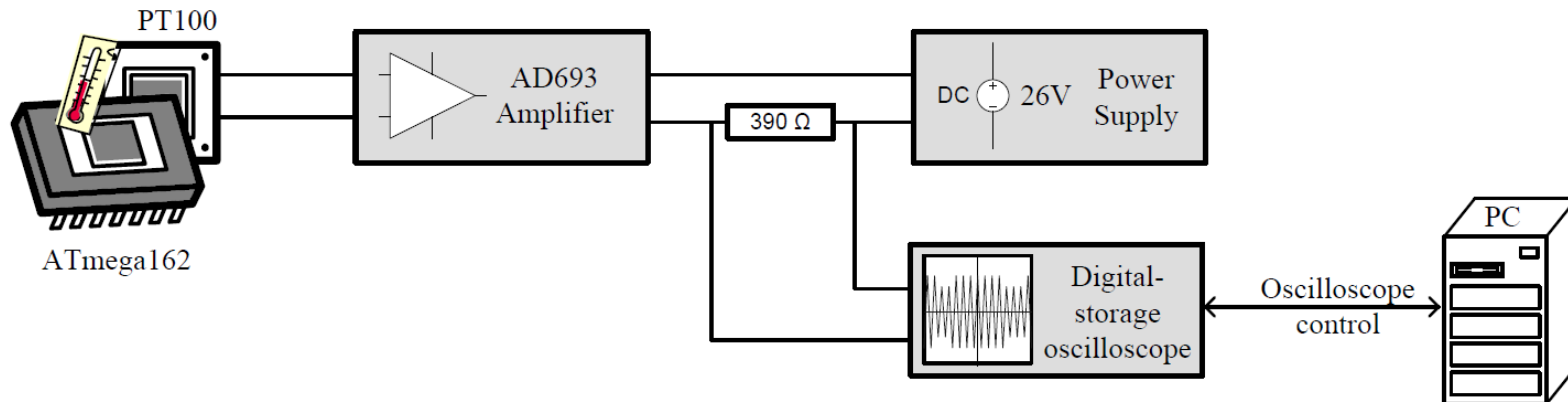
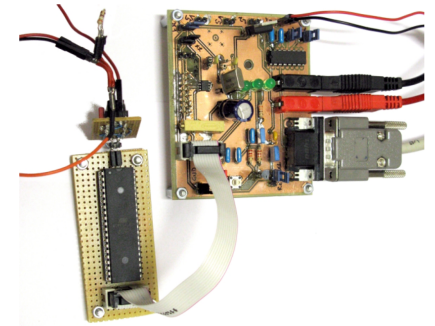
Example of How Power Spike will Tamper with a Loop Bound

- Corrupted PC could lead to infinite loop
- Branch target address corruption causes incorrect loop bound



Experimental Setup for Temperature Attack

- Target: ATmega162 AVR microcontroller
- Temperature measuring through PT100
- AD693 amplifier for accurate measurements (0~140°C)
- PC controls measuring process (e.g. MATLAB)



Bibliography

- [J. Balasch, FTDC'11] J. Balasch, B. Gierlichs and I. Verbauwhede, "An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs," *FDTC*, pp. 105-114, 2011
- [M. Arora, online, July 2012] https://www.eetimes.com/document.asp?doc_id=1279619
- [Seagate, Technology paper, 2008] <http://dator8.info/pdf/AES/3.pdf>
- [D. Karaklajic, TVLSI'13] D. Karaklajic, J.-M. Schmidt, and I. Verbauwhede, "Hardware designer's guide to fault attacks", *TVLSI*, vol. 21, no. 12, pp. 2295-2306, 2013
- [R. Baumann, IEEE Design & Test of Computers'05] R. Baumann, "Soft errors in advanced computer systems," , *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258-266, 2005
- [S. Kumar, FDTC'12] Santosh Kumar, Shalu Agarwal, Jae Pil Jung, "Soft Error Issue And Importance of Low Alpha Solders for Microelectronics Packaging," *Reviews on advanced materials science*, vol. 34, no. 2, pp. 185-202, 2013
- [M. J. Gadlage, TNS'11] M. J. Gadlage et al., "Alpha-Particle and Focused-Ion-Beam-Induced Single-Event Transient Measurements in a Bulk 65-nm CMOS Technology," in *IEEE Transactions on Nuclear Science*, vol. 58, no. 3, pp. 1093-1097, June 2011
- [B. Narasimham, TNS'07] B. Narasimham *et al.*, "Characterization of Digital Single Event Transient Pulse-Widths in 130-nm and 90-nm CMOS Technologies," in *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2506-2511, Dec. 2007
- [S. Guilley, Springer'12] S. Guilley and J.-L. Danger, *Global Faults on Cryptographic Circuits*, Chapter 17 in *Fault Analysis in Cryptography*, Marc Joye and Michael Tunstall (Ed.), Springer, pp. 295-311, 2012
- [M. Agoyan, PASTIS'10] Michel Agoyan, Jean-Max Dutertre Dutertre, Amir-Pasha Mirbaha Mirbaha, David , David Naccache Naccache, Anne-Lise Ribotta Ribotta and Assia Tria, *Single-Byte Laser Faults using Large Spots 2nd PAcA Security Trends In embedded Security workshop (PASTIS'2010)*, Gardanne FRANCE 16-17 June 2010. online https://www.emse.fr/~dutertre/doc_recherche/C_2010_1_PASTIS2010.pdf
- [J. Balasch, CT-RSA'12] J. Balasch, B. Gierlichs, R. Verdult, L. Batina, I. Verbauwhede, "Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs", *CT-RSA*, vol. 7178, pp. 19-34, 2012

Bibliography

- [M. Hutter, CHES'08] M. Hutter, J.-M. Schmidt, T.Plos, "RFID and its Vulnerability to Faults," CHES, vol. 5154, pp. 363-379, 2008
- [M. Hutter, CARDIS'13] M. Hutter, J.-M. Schmidt, "The Temperature Side Channel and Heating Fault Attacks", CARDIS, 2013
- [Woudenberg, FDTC'11] J. G. J. van Woudenberg, M. F. Witteman and F. Menarini, "Practical Optical Fault Injection on Secure Microcontrollers," *FDTC*, pp. 91-99, 2011
- [S. Skorobogatov, HWA'11] Sergei Skorobogatov, Physical Attacks on Tamper Resistance: Progress and Lessons, 2nd ARO Special Workshop on HW Assurance, 2011
- [D. Naccache, IACR'04] David Naccache, Phong Q. Nguyen, Michael Tunstall, and Claire Whelan, "Experimenting with Faults, Lattices and the DSA", IACR, pp 16-28, 2004
- [S. Skorobogatov, CHES'02] S. Skorobogatov and R. Anderson, "Optical Fault Induction Attacks," CHES, pp. 2-12, 2002
- [S. Skorobogatov, FDTC'10] S. Skorobogatov, "Optical Fault Masking Attacks," *FDTC*, pp. 23-29, 2010
- [B. Giller, BlackHat'15] B. Giller, "Implementing Practical Electrical Glitching Attacks," Black Hat Europe, 2015, online: <https://www.blackhat.com/docs/eu-15/materials/eu-15-Giller-Implementing-Electrical-Glitching-Attacks.pdf>
- [N. Selmane, EDCC'08] N. Selmane, S. Guilley and J. L. Danger, "Practical Setup Time Violation Attacks on AES," Seventh European Dependable Computing Conference, Kaunas, pp. 91-96, 2008
- [Kim C.H., WISTP'07] Kim C.H., Quisquater JJ. Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In: Sauveron D., Markantonakis K., Bilas A., Quisquater JJ. (eds) Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, WISTP, pp. 215-228, 2007
- [H. Pahlevanzadeh, PhD thesis, 2015] H. H. Pahlevanzadeh, Assessing and Improving the Reliability and Security of Circuits Affected by Natural and Intentional Faults, UNH PhD thesis, 2016.

A Whitebox Introduction to Fault Attacks

Patrick Schaumont

schaum@vt.edu

Professor

Karine Heydemann

karine.heydemann@lip6.fr

Associate Professor

Qiaoyan Yu

qiaoyan.yu@unh.edu

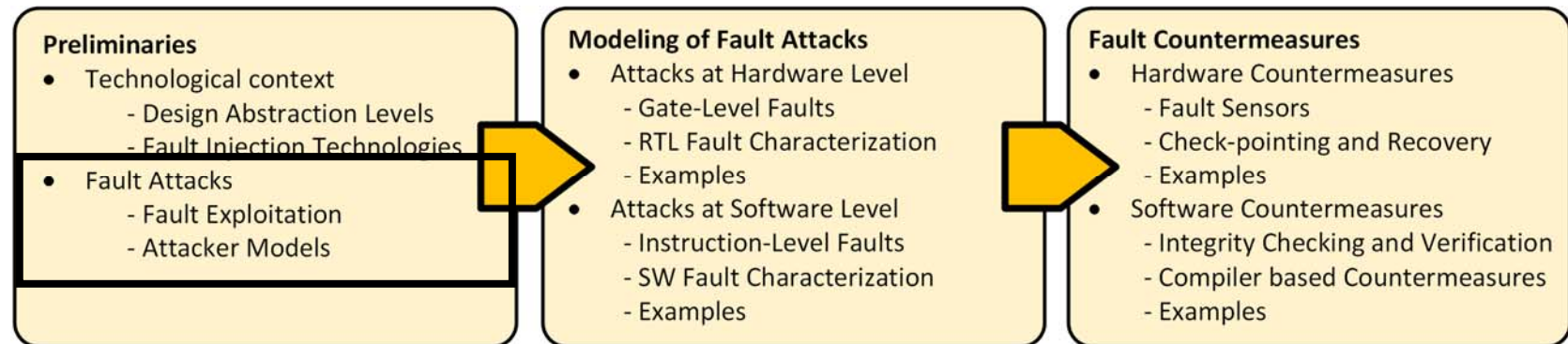
Associate Professor



Part 1: Preliminaries

Fault Attacks

Patrick Schaumont



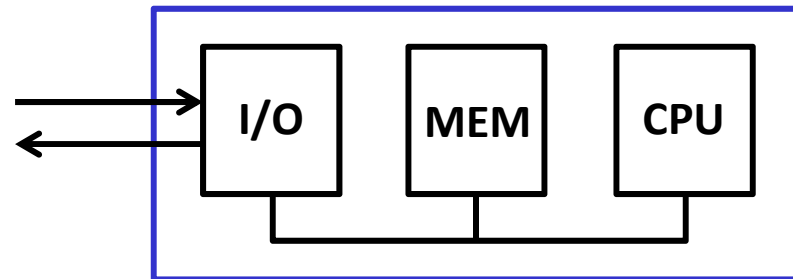
Outline

1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Common Fault Exploitation Techniques

Outline

1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Common Fault Exploitation Techniques

Attacks on Embedded Software




- Embedded Software assumes execution is correct
- **Incorrect execution** as starting point for attack
 - Privilege Escalation
 - Information Leakage

Escalation & Leakage

- **Privilege Escalation**
= Adversarial Control of Critical Decisions

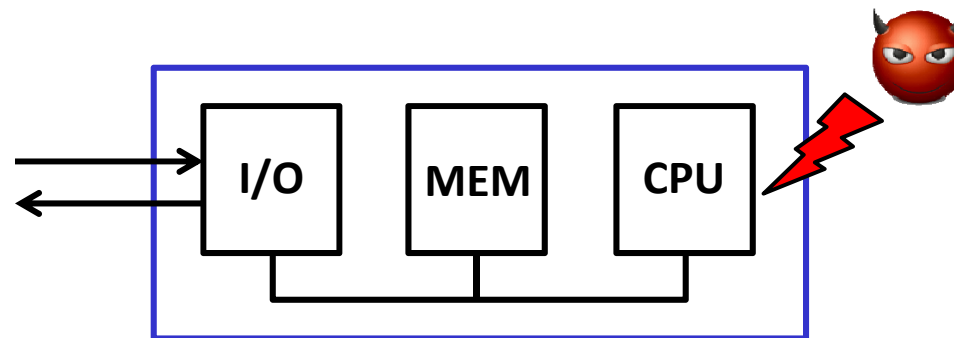
```
if (! access_allowed )  
    abort( );
```

- **Information Leakage**
= Disclosure of Secret Data & Dependencies

```
 r1 if (key_bit)  
        out = f(r1);  
else  
        out = f(r0);
```

key_bit leaks through out

Triggering *Incorrect Execution*



Attacker	Attack Target	Security Failure
Input/Output Attacker	Input/Output Data	Software Bugs
Memory Attacker	Application/Task Image	Lack of Mem Isolation
Hardware Attacker	Instruction	Opcode Modification
	Instruction Execution	Micro-Architecture
	Circuit	Timing, Threshold Levels
	Environment	Operating Conditions

this tutorial

Outline

1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Common Fault Exploitation Techniques

Anatomy of a Fault Attack



1. Fault Attack Design

- Fault Target and Fault Model
- Fault Injection Method
- Fault Exploitation Method

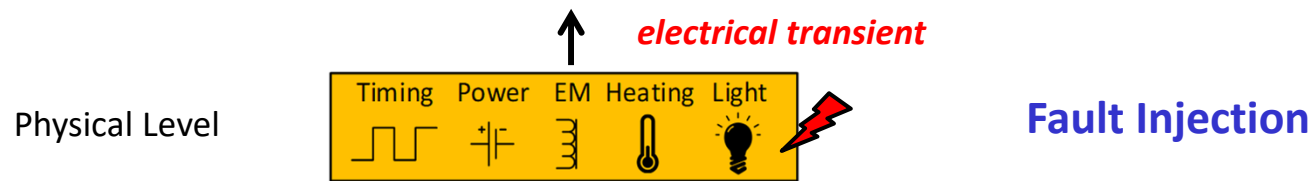
**Defined by
Security (Attack)
Objective**

2. Fault Attack Implementation

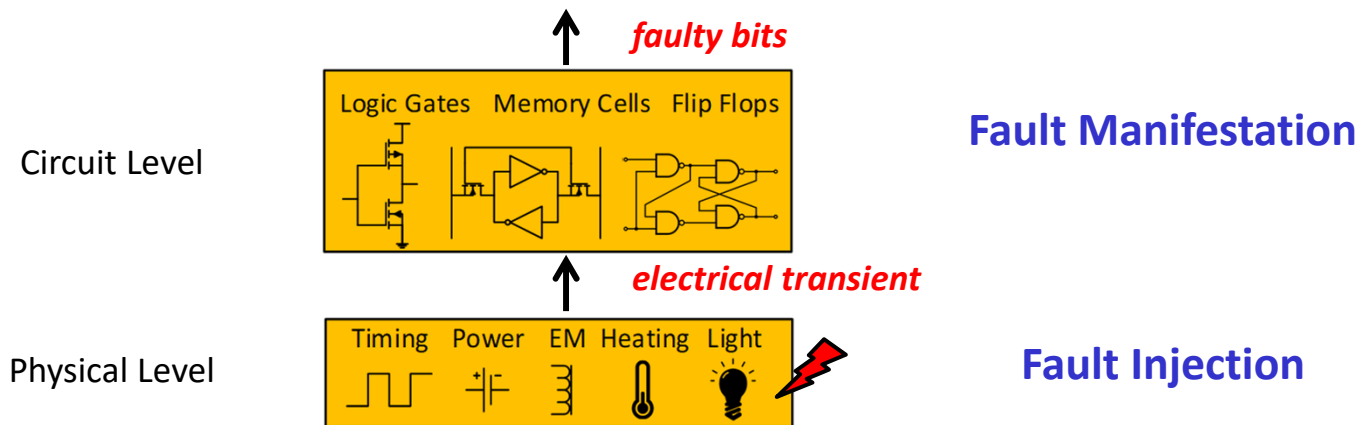
- Fault Injection
- Fault Manifestation
- Fault Propagation
- Fault Observation
- Fault Exploitation

**Constrained by
Implementation**

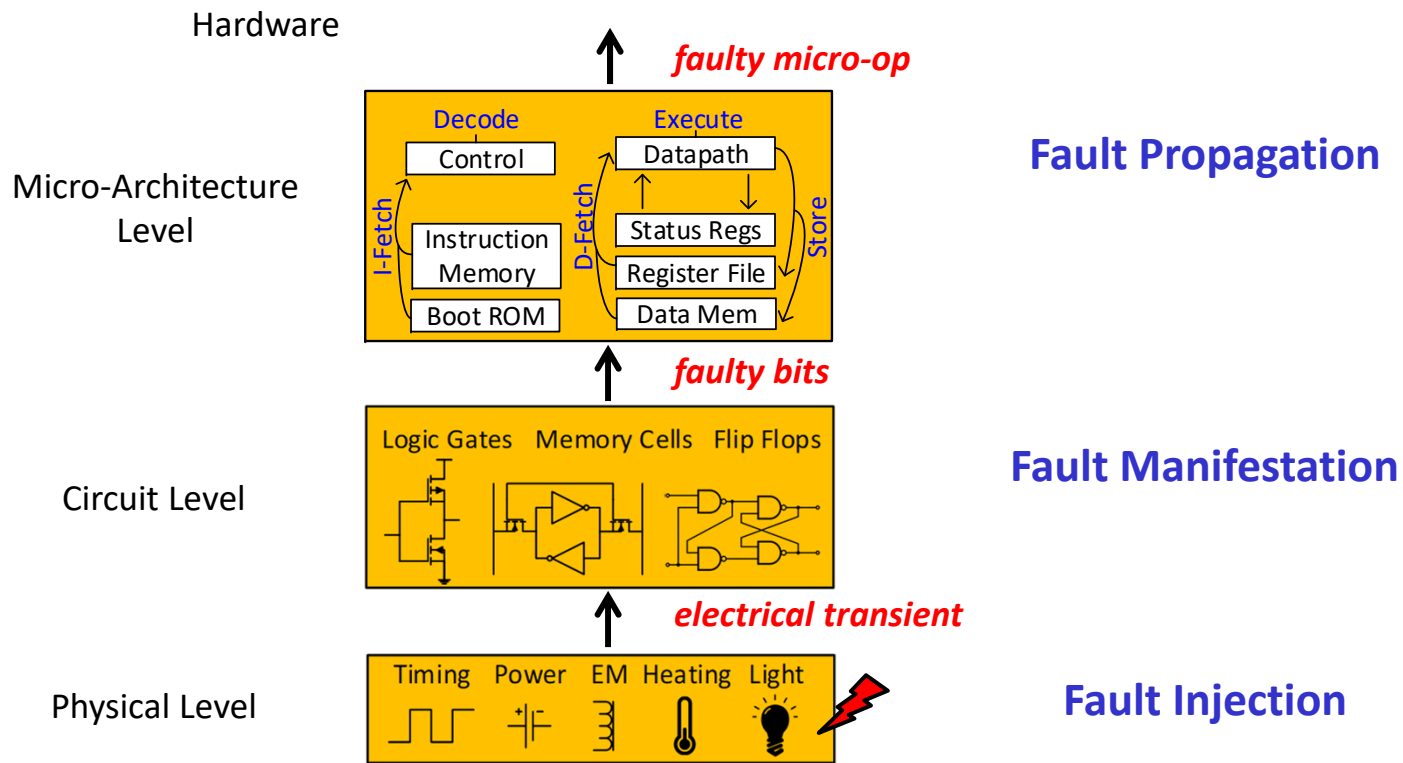
Anatomy of a Fault Attack



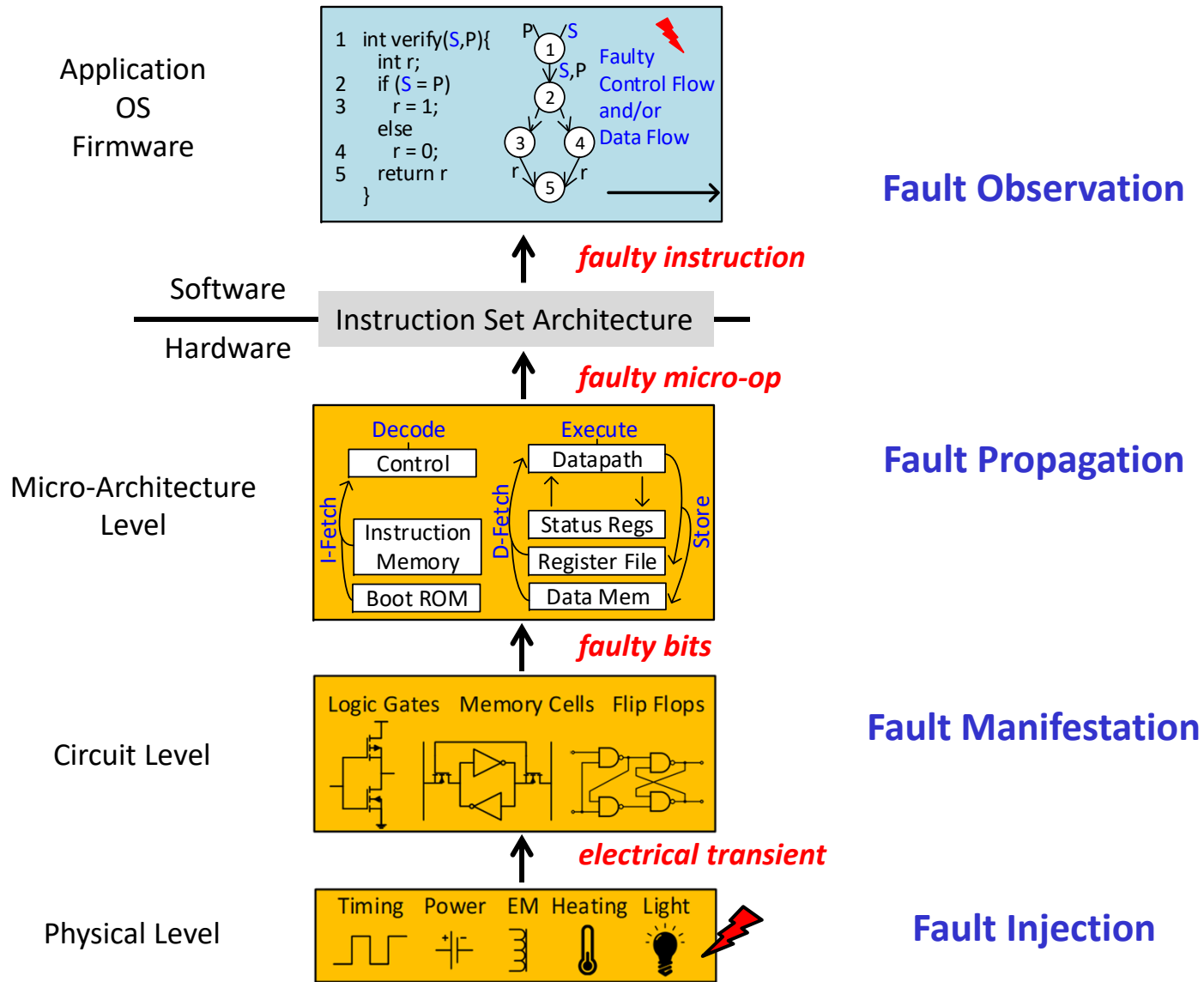
Anatomy of a Fault Attack



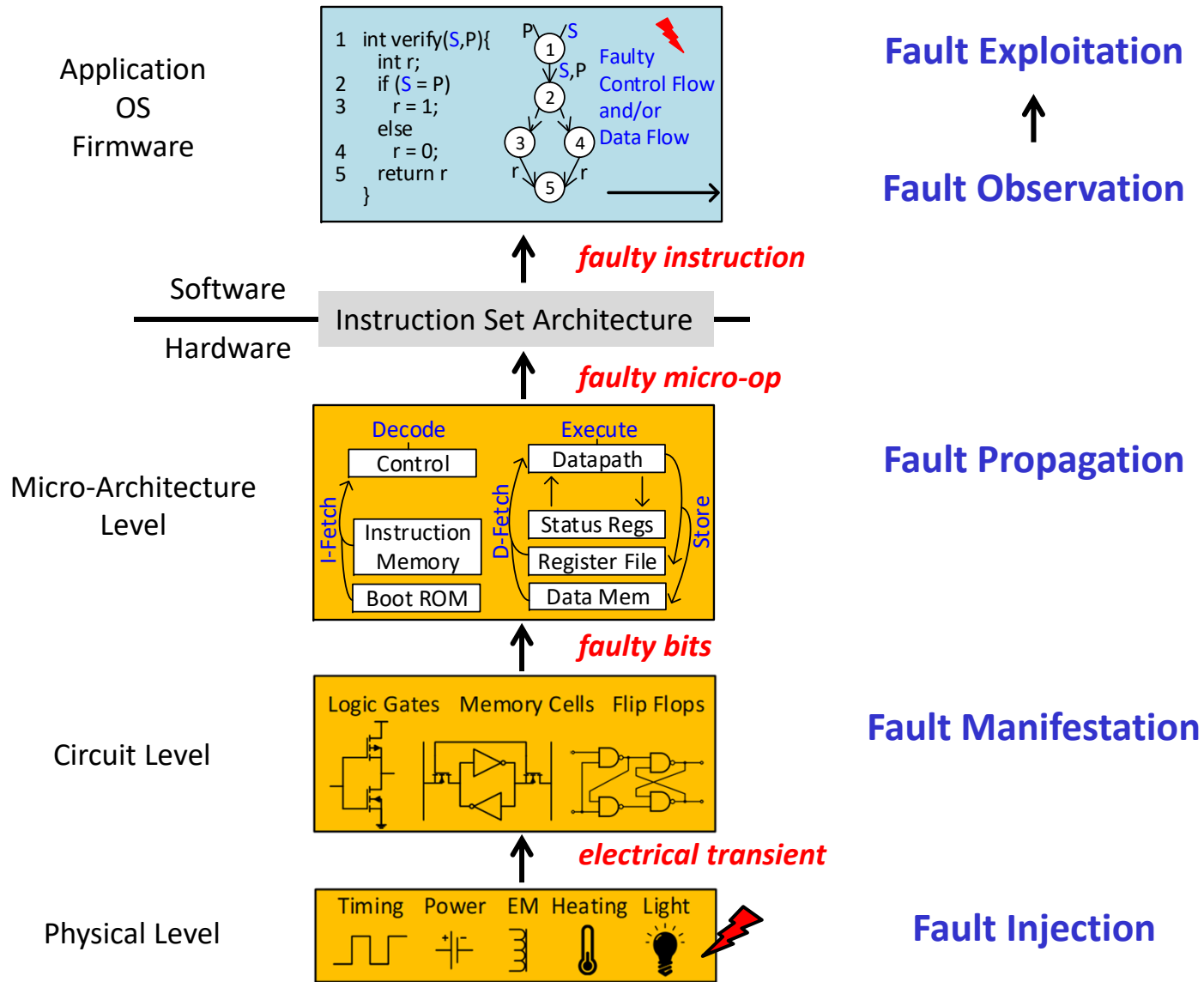
Anatomy of a Fault Attack



Anatomy of a Fault Attack



Anatomy of a Fault Attack



Outline

1. Introducing the Fault Attack
2. Anatomy of a Fault Attack
3. Common Fault Exploitation Techniques

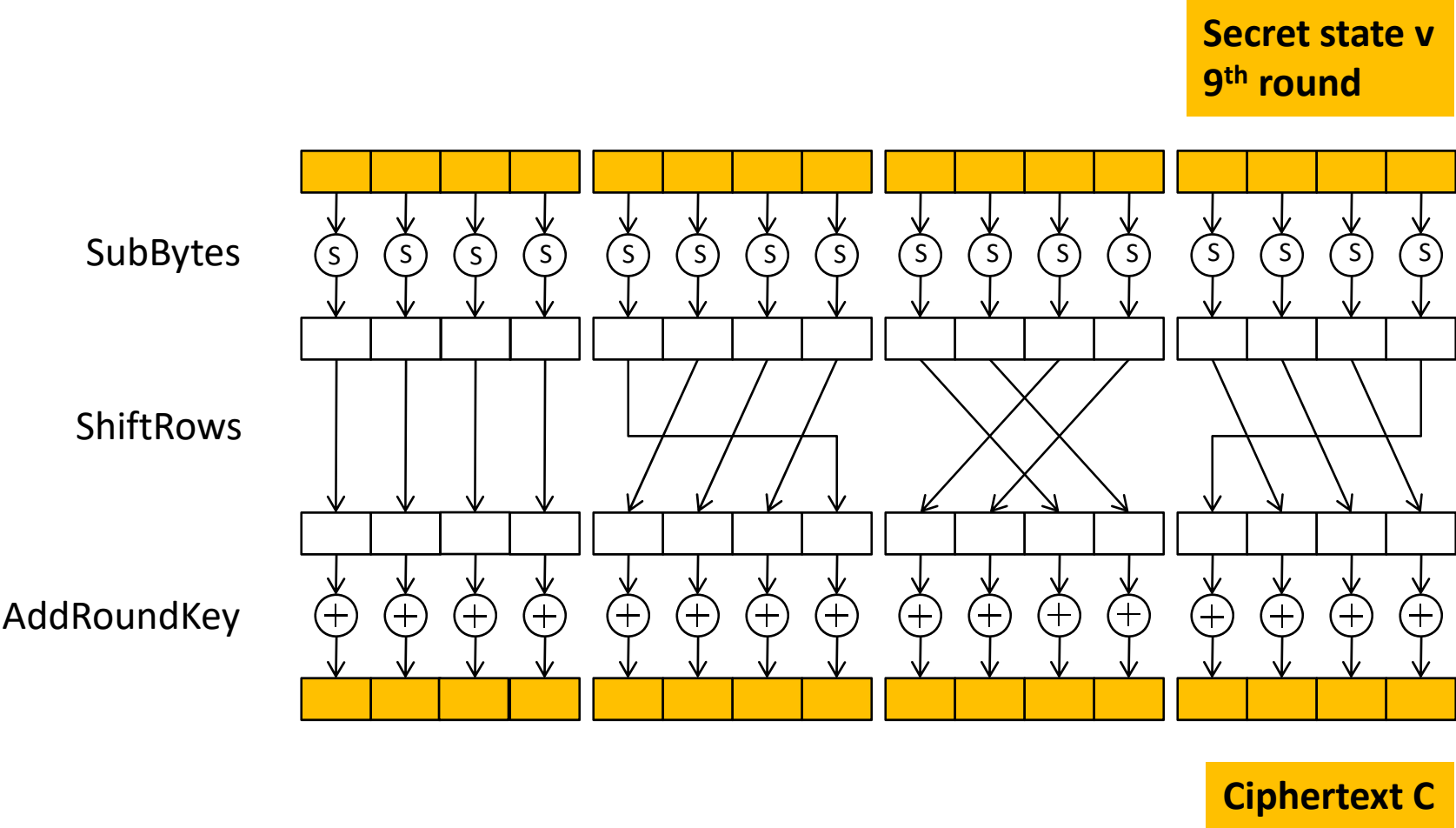
Common Fault Exploitation

- Cryptanalysis using fault injection
 - Differential Fault Analysis
 - Biased Fault Analysis
 - Safe Error Analysis
 - Algorithm-specific Fault Analysis
- Fault-aided Side-channel Analysis
- Fault-enabled Logical Attacks
- Fault-aided Reverse Engineering


Common Fault Exploitation

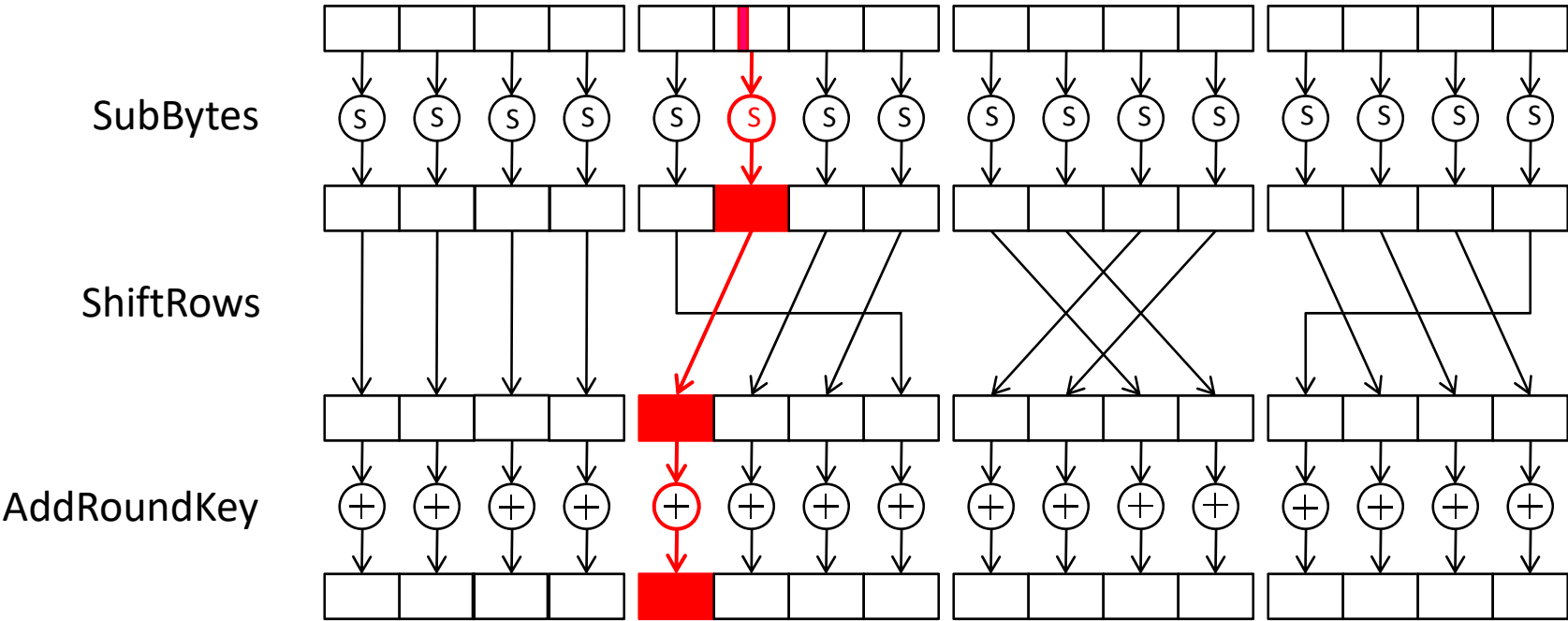
- **Cryptanalysis using fault injection**
 - **Differential Fault Analysis**
 - **Biased Fault Analysis**
 - **Safe Error Analysis**
 - Algorithm-specific Fault Analysis
- Fault-aided Side-channel Analysis
- **Fault-enabled Logical Attacks**
- Fault-aided Reverse Engineering

Differential Fault Analysis



Bit-flip Attack on AES

 **Fault Model:**
Bit-flip on a secret state bit



A bit-flip results in a faulty ciphertext byte

Bit-flip Attack on AES

- **Fault Differential**

$$c = \text{sbox}(v) \oplus k$$

$$c' = \text{sbox}(v') \oplus k$$

$$\text{Hence } \Delta = c \oplus c' = \text{sbox}(v) \oplus \text{sbox}(v')$$

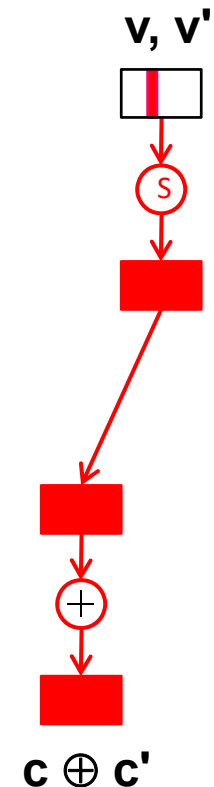
- **Fault Analysis**

Reconstruct v by analyzing Δ

Once we know v , we find the last round-key as:

$$k = \text{sbox}(v) \oplus c$$

32 bit-flip faults in round 10 disclose entire key



Classic DFA



[Tunstall 2010] Single random byte fault at 8th round of AES-128: Key $2^{128} \rightarrow 2^{12}$

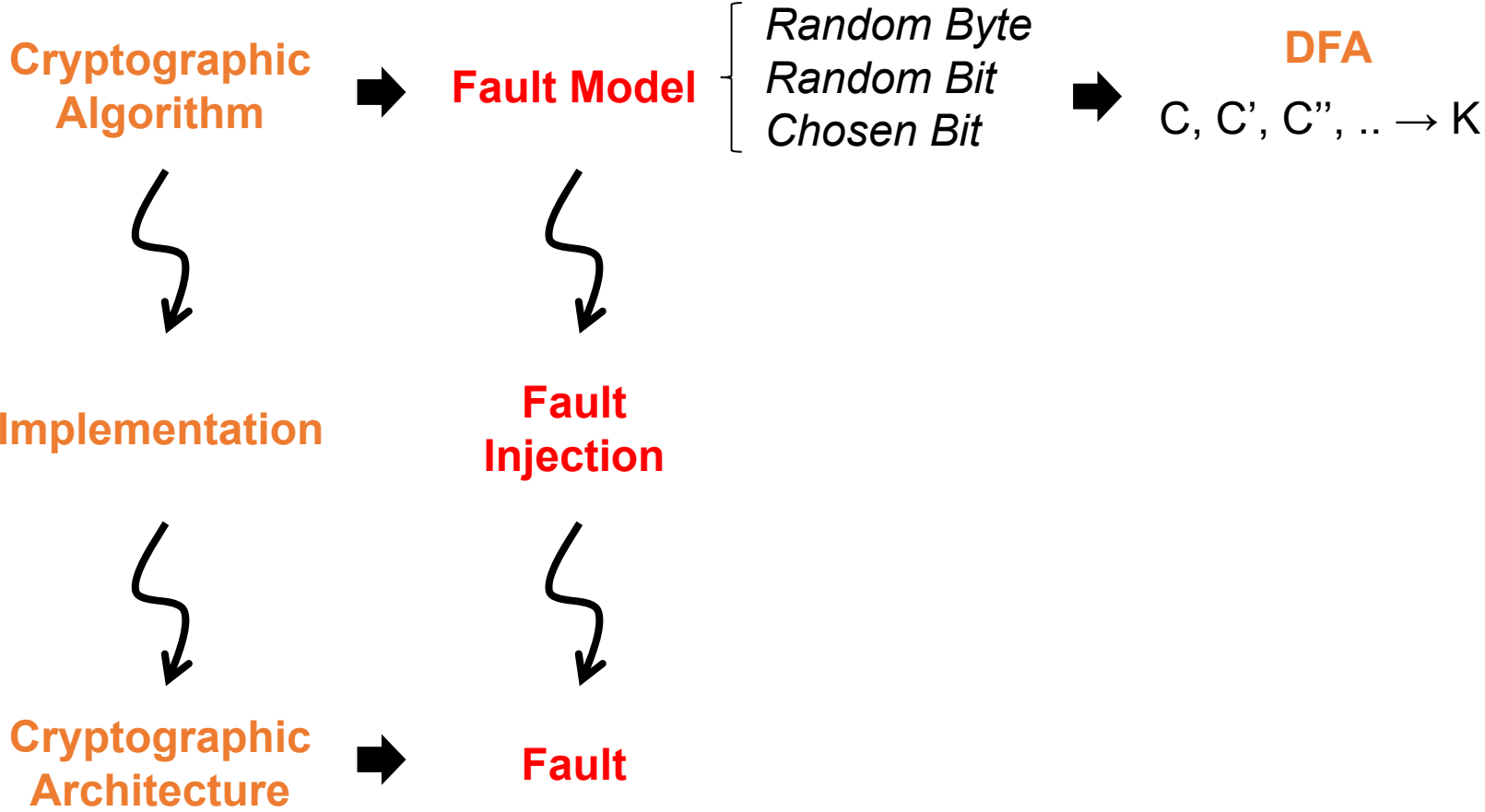
[Ali 2012] Two seq. byte fault at 9th, 10th round of AES-192: Key $2^{128} \rightarrow 1$

Current DFA methods are *optimal*

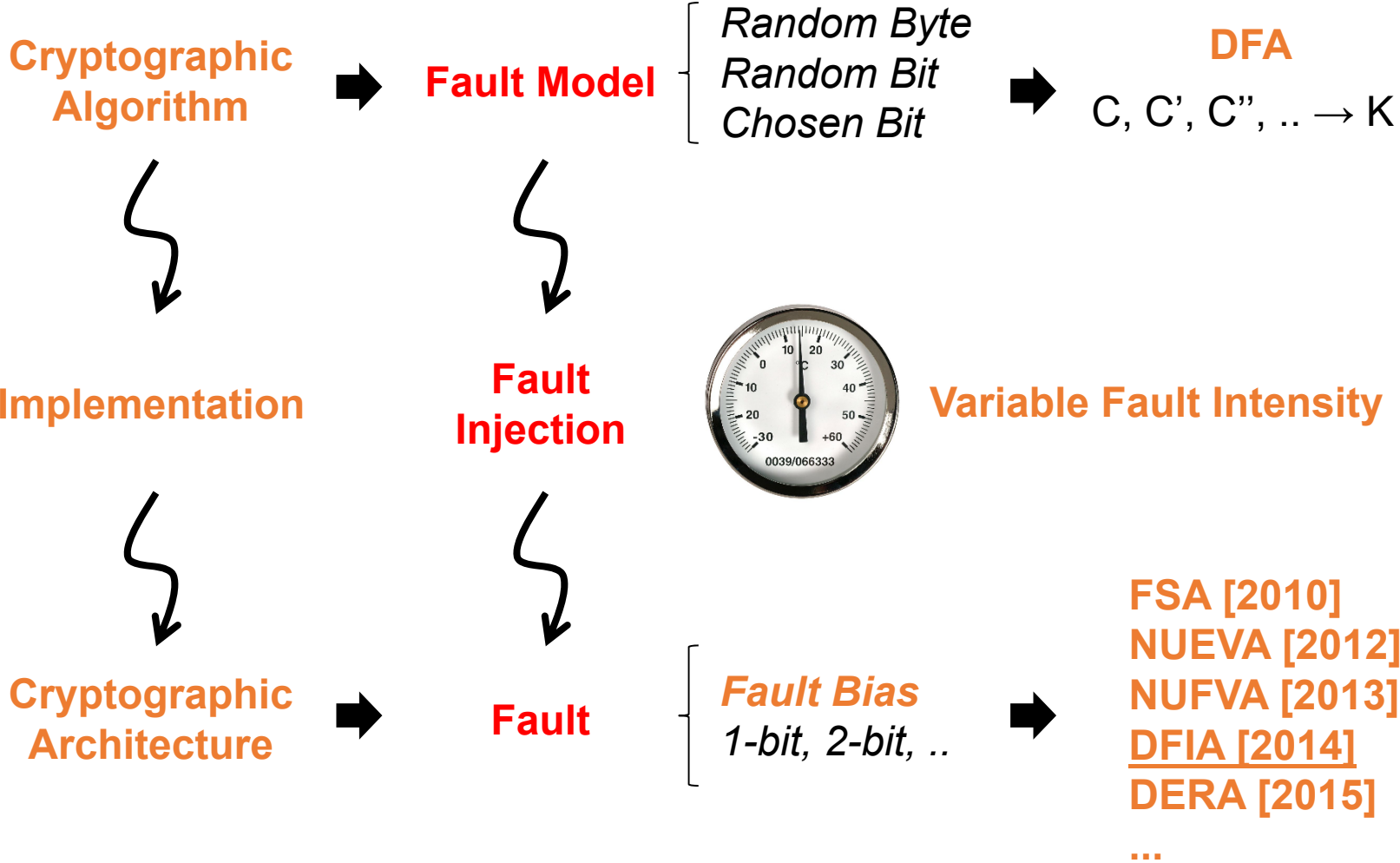
IF

the fault model can be realized

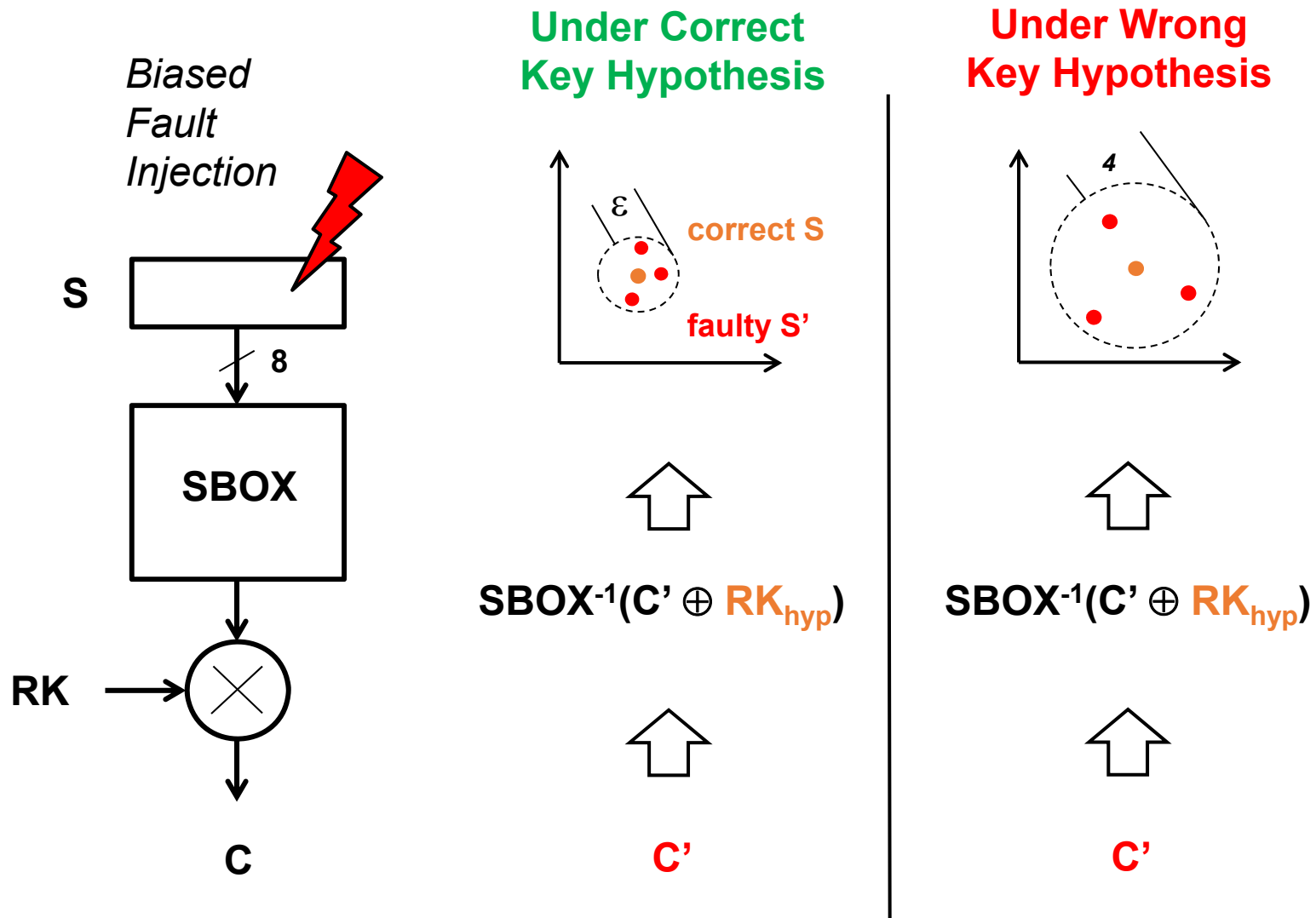
Biased Fault Analysis



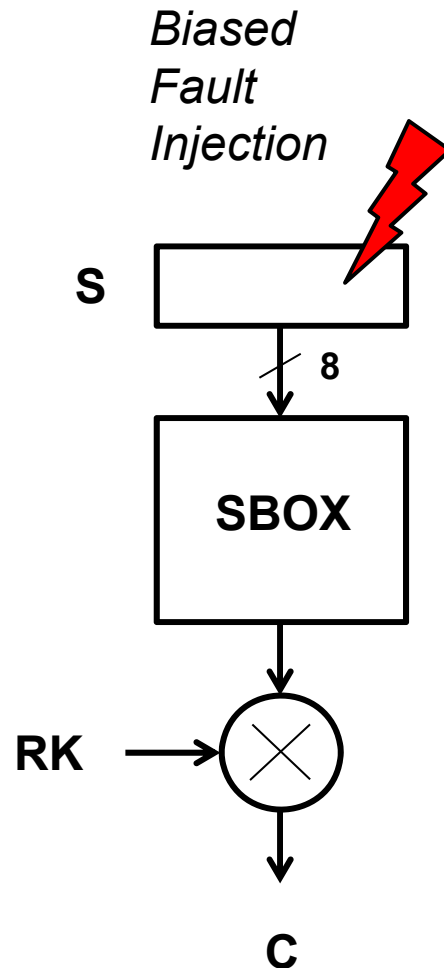
Biased Fault Analysis



Biased Fault Analysis



Differential Fault Intensity Analysis



Differential Fault Intensity Analysis

1. Inject Faults at different Fault Intensities
 $\text{HW}(S \oplus S') < \varepsilon$
2. Collect Fault Ciphertext C'
3. For all Key hypothesis RK_{hyp} compute
 $S_{i,\text{RK}} = \text{SBOX}^{-1}(C' \oplus \text{RK}_{\text{hyp}})$
4. Select RK for which

$$\text{RK} = \text{ArgMin}(\sum_i \sum_j \text{HD}(S_{i,\text{RK}}, S_{j,\text{RK}}))$$

Safe-error Analysis

Input: Elliptic Curve Point P
secret integer k

Output: $k.P$

1. $R[0] = 0$

2. for $i = 1 - 2$ down to 0 do

3. $R[0] = 2.R[0]$

4. $R[1] = R[0] + P$

5. $R[0] = R[k_i]$

6. end for

Return $R[0]$

**Double-Add Always
(SPA Countermeasure)**

Safe-error Analysis

Input: Elliptic Curve Point P
secret integer k

Output: $k.P$

1. $R[0] = 0$

2. for $i = 1 - 2$ down to 0 do

3. $R[0] = 2.R[0]$

4. $R[1] = R[0] + P$

5. $R[0] = R[k_i]$

6. end for

Return $R[0]$

**Every loop iteration,
one of these is
a dummy operation**

Safe-error Analysis

Input: Elliptic Curve Point P
secret integer k

Output: $k.P$

1. $R[0] = 0$

2. for $i = 1 - 2$ down to 0 do

3. $R[0] = 2.R[0]$

4. **$R[1] = R[0] + P$**

5. $R[0] = R[k_i]$

6. end for

Return $R[0]$

C-safe error:

Injecting a fault in a dummy operation will not affect the output

Fault Enabled Logical Attacks

- General-purpose computing

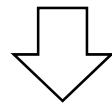
- Memory dump extraction
- Control-flow hijacking
- Privilege escalation
- Secure Boot bypass
- Memory disturbance error attacks
- DVFS interface attacks

Hardware-Controlled

Software-Controlled

Memory Dump Attack

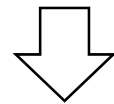
```
for (i = 0; i < len; i++)  
    output = buffer[i];
```



```
    ld        [ %i0 + %g1 ], %g3  
label:  
    st        %g3, [ %g2 ]  
    add       %g1, 4, %g1  
    cmp       %g1, 0x40  
    bne,a    label  
    ld        [ %i0 + %g1 ], %g3  
    ret
```

Memory Dump Attack

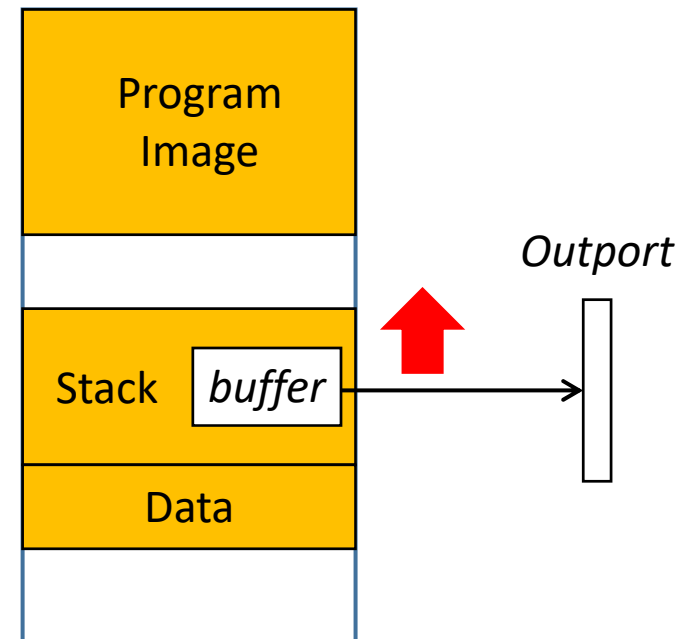
```
for (i = 0; i < len; i++)  
    outport = buffer[i];
```



```
ld      [ %i0 + %g1 ], %g3  
label:  
st      %g3, [ %g2 ]  
add     %g1, 4, %g1  
cmp     %g1, 0x40  
bne, a label      Instruction-skip  
ld      [ %i0 + %g1 ], %g3  
ret
```



Memory Map



Buffer Overflow Attack

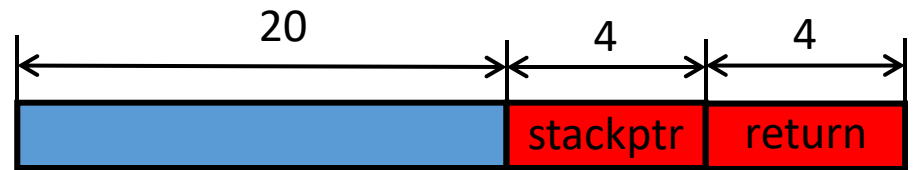
ARM Cortex M0

```
void myfunc(char *buf) {  
    char msg[20] = {0};  
    strncpy(msg, buf, sizeof(msg) - 1);  
    ..  
}
```

Shoei Nashimoto, Naofumi Homma, Yu-ichi Hayashi, Junko Takahashi, Hitoshi Fuji, Takafumi Aoki: Buffer overflow attack with multiple fault injection and a proven countermeasure. *J. Cryptographic Engineering* 7(1): 35-46 (2017)


Buffer Overflow Attack

```
void myfunc(char *buf) {  
    char msg[20] = {0};  
    strncpy(msg, buf, sizeof(msg) - 1);  
    ..  
}
```



```
void *memcpy (void *dest,  
             const void *src,  
             size_t len) {
```

malicious buf



```
    char *d = dest;  
    const char *s = src;  
    while (len--) Instruction-skip  
        *d++ = *s++;  
    return dest;  
}
```

Privilege Escalation

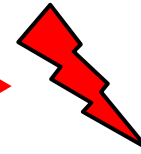
ARM Cortex A-9

```
r0 = 0 .. r11 = 0
r7 = 0xd0
svc #0          // setuid system call
if (r0 == 0) // success
    system("/bin/sh");
```

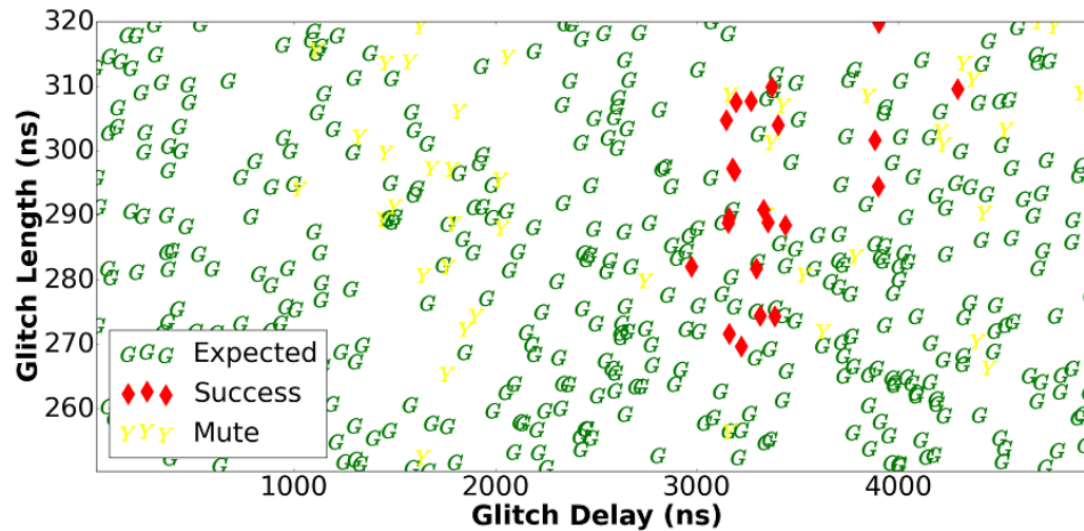
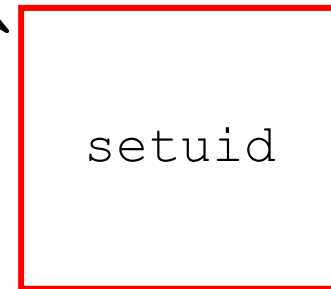
Privilege Escalation

```
r0 = 0 .. r11 = 0
r7 = 0xd0
svc #0 // setuid system call
if (r0 == 0) // success
    system("/bin/sh");
```

ARM Cortex A-9

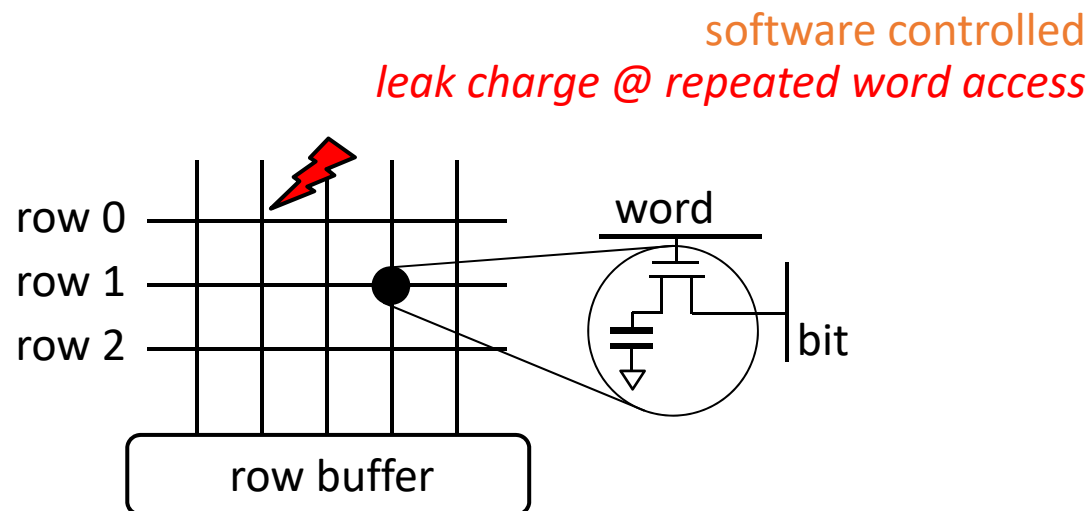


System Call



- 18968 experiments
- 21 hours
- 1.3% success rate
- Root Shell spawned every 5 minutes

Memory Disturbance Error Attack



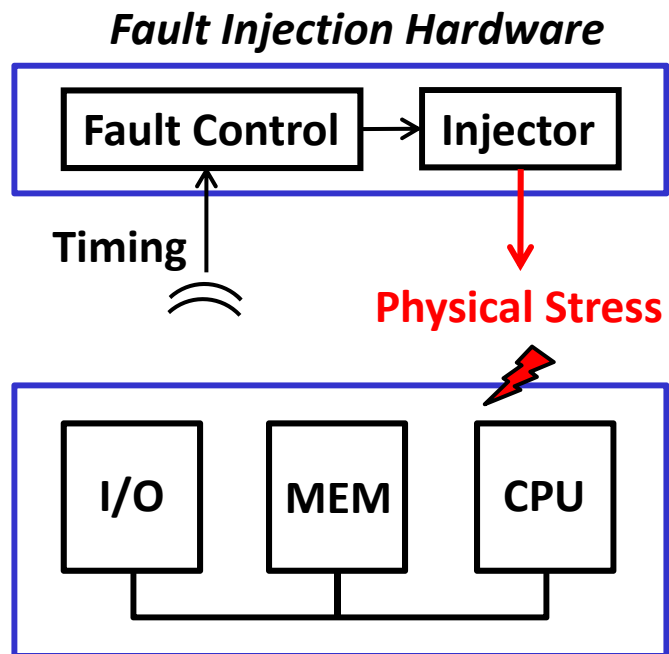
- Sandbox escape and Memory Access Privilege Escalation [Seaborn 2015]
- Bit flip achieved in a cloud setting [Razavi 2016]
- Bit flip achieved through Javascript [Gruss 2016]
- Bit flip achieved through Android [Van der Veen 2016]

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. ISCA 2014: 361-372

The present and the Future

Hardware-controlled Fault Injection

1997 (Bellcore) - now



The present and the Future

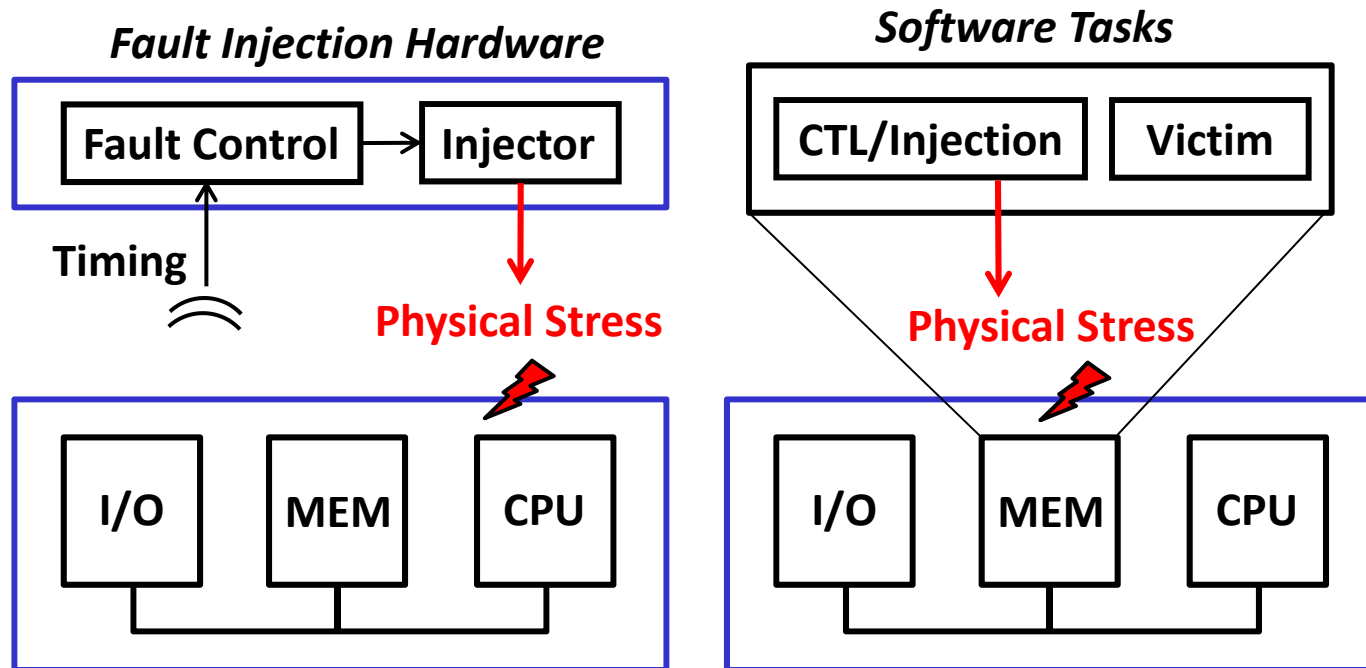
**Hardware-controlled
Fault Injection**

1997 (Bellcore) - now



**Software-controlled
Fault Injection**

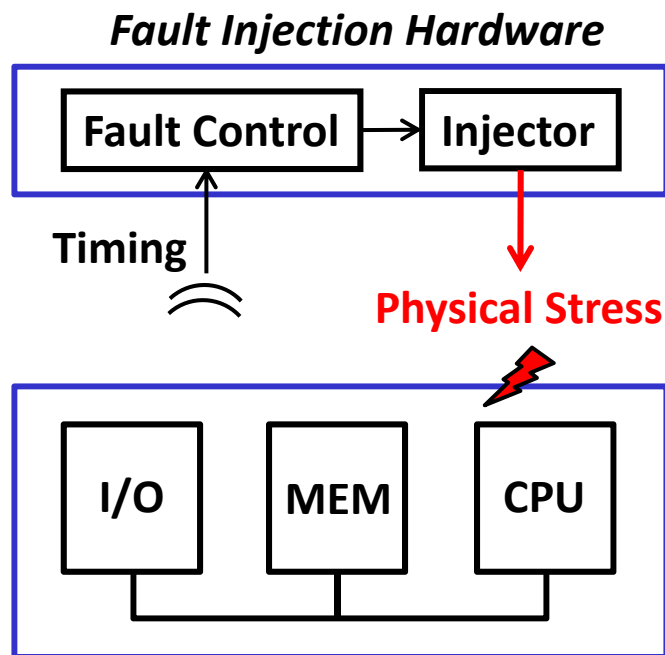
2014 (Rowhammer) - now



The present and the Future

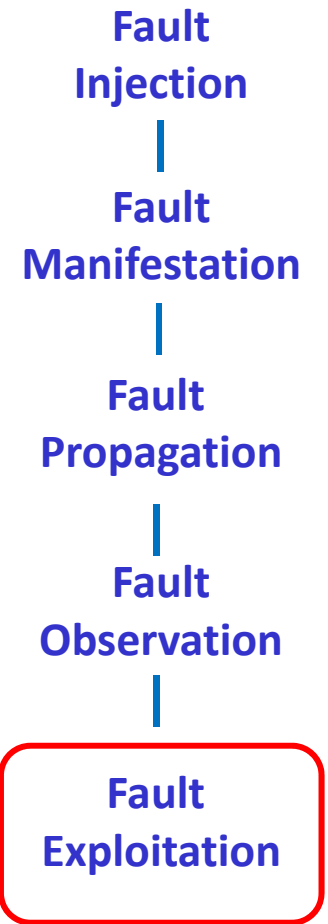
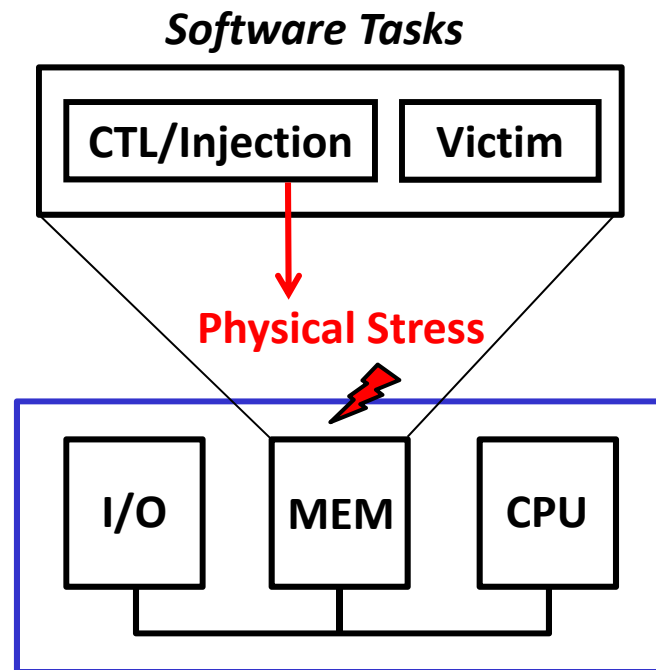
Hardware-controlled Fault Injection

1997 (Bellcore) - now



Software-controlled Fault Injection

2014 (Rowhammer) - now



Part 2: Modeling of Fault Attacks

---Attacks at Hardware Level

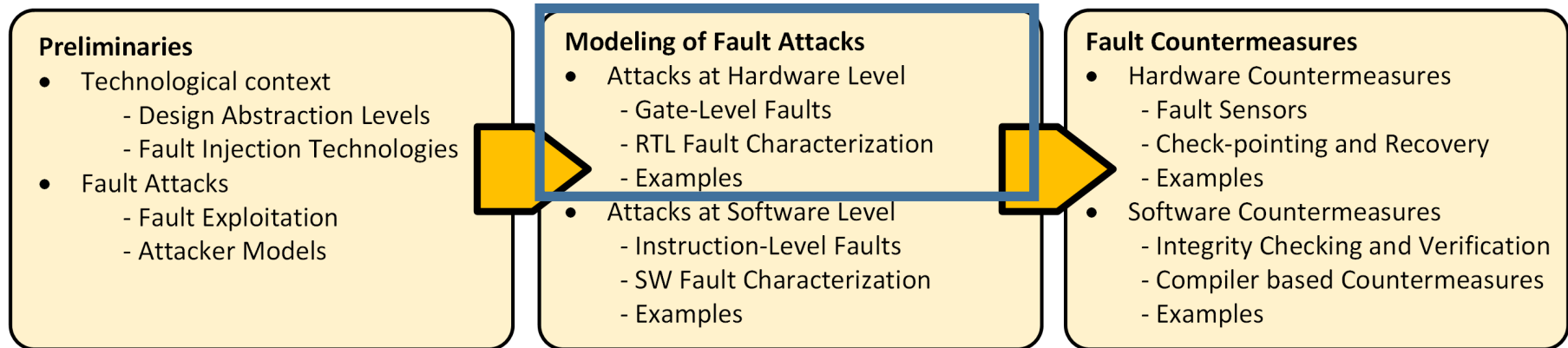
Qiaoyan Yu

qiaoyan.yu@unh.edu

Associate Professor



**University of
New Hampshire**

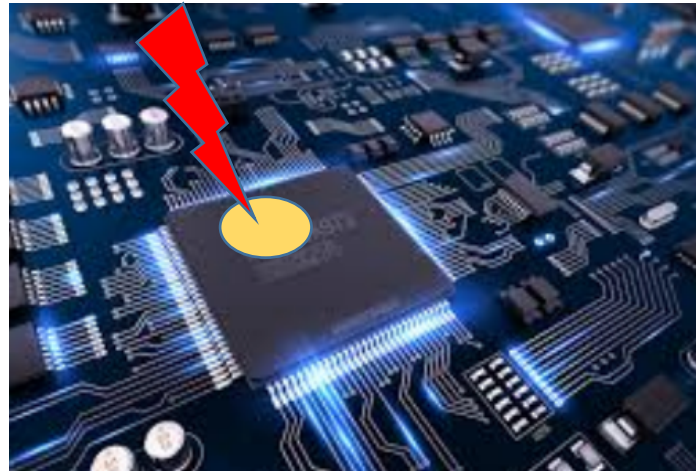


Outline

- Introduction of fault model
- Physical mechanisms exploited by fault attacks
- Fault modeling
 - Transistor level
 - Gate level
 - Register transfer level (RTL)
- Fault Characterization
 - Overclocking attack
 - Under-powering attack
- Tools for fault injection and analysis

Assumptions Constitute the Fault Model

- Location of the fault in the circuit
- Precise time of fault injection
- Specific value of a faulty variable



Impact of Fault Injection Location on Key Retrieval

Number of faulty ciphertexts

Fault injection locations

250



Computation of 9th round^[1]

128 – 256



Computation^[2]

40



A byte between computation of 8th and 9th round MixColumn^[3]

2



Input for 8th or 9th round^[4]

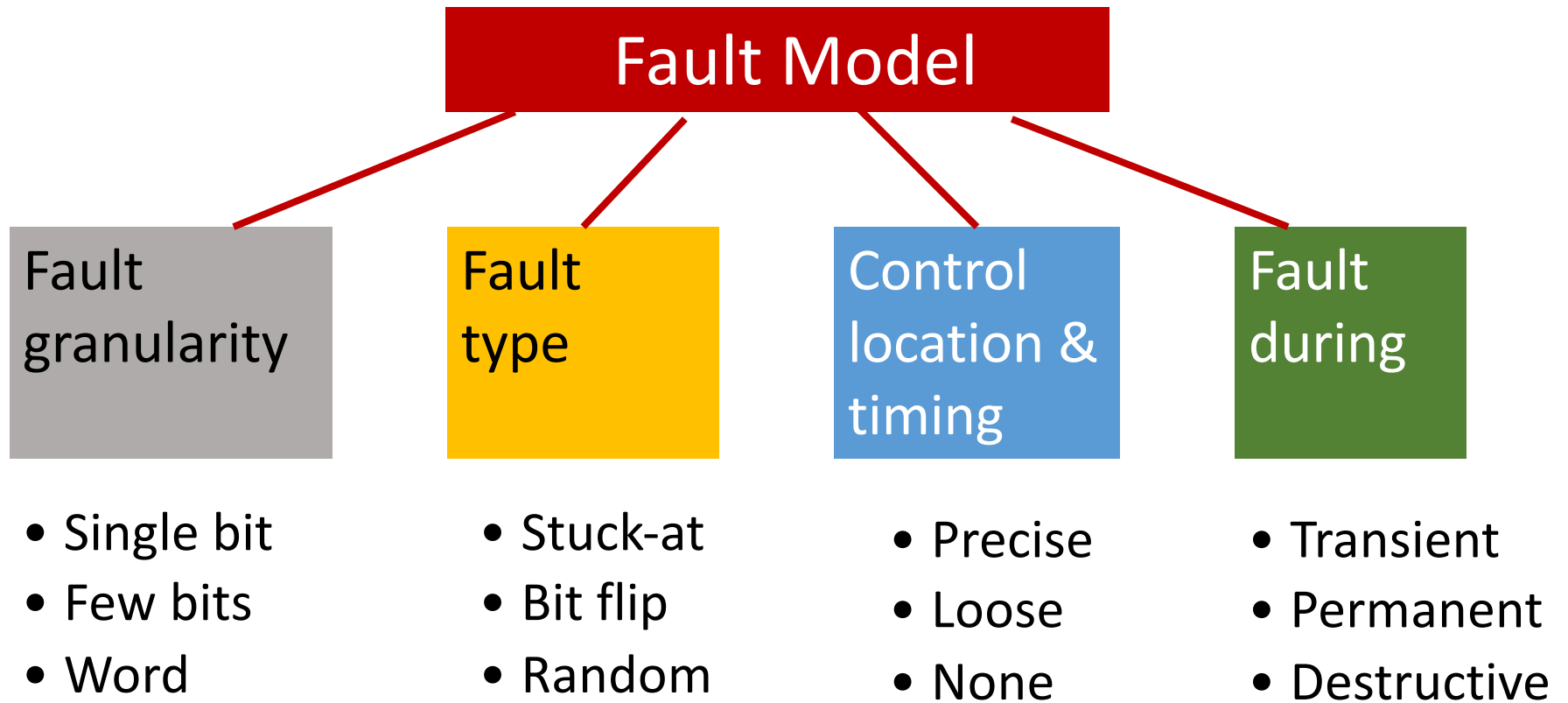
[1] C. Giraud, LNCS'04.

[2] J. Blömer, LNCS'03.

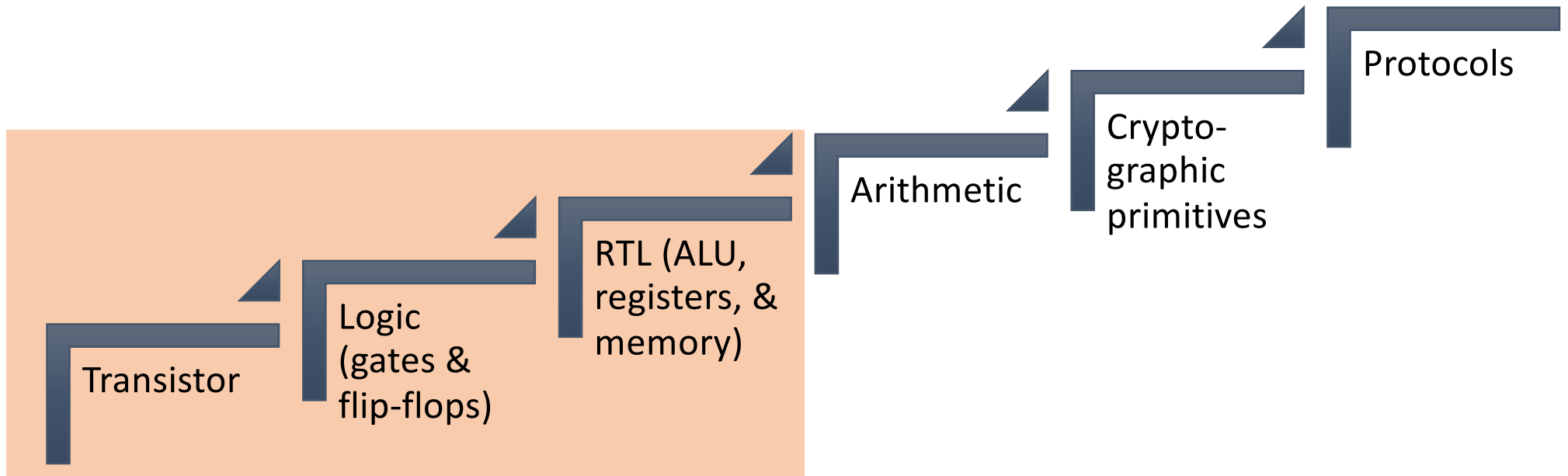
[3] P. Dusart, LNCS'03.

[4] G. Piret, CHES'03.

Four Aspects of a Fault Model

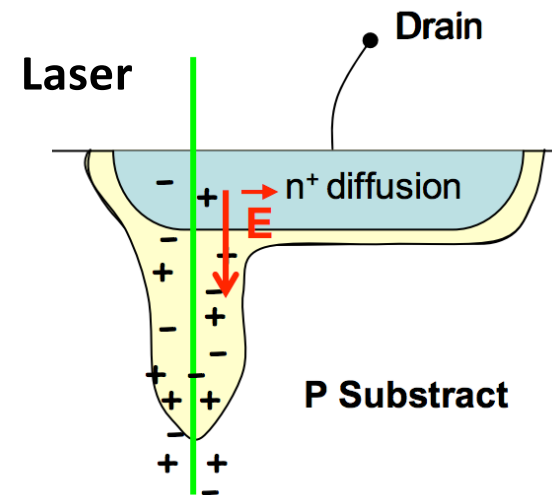


Abstract Levels in Cryptographic Hardware Design Process



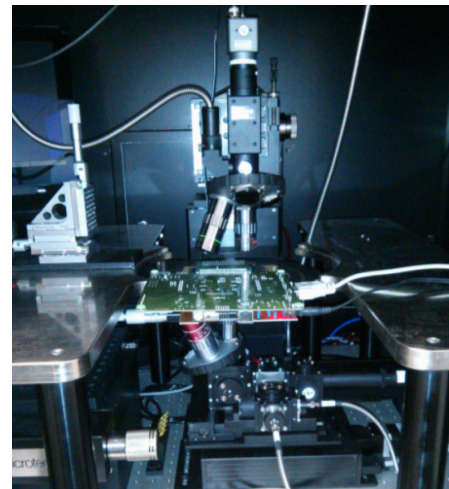
Laser-based Fault Attacks on VLSI

- Laser
 - Light Amplification by Stimulated Emission of Radiation
 - Good spatial and temporal precision precision to inject the faults
 - Diameter of laser beam
 - Wavelength
 - Amount of emitted energy
 - Impact coordinates (attacked circuit part)
 - Impact moment
 - Exposure duration

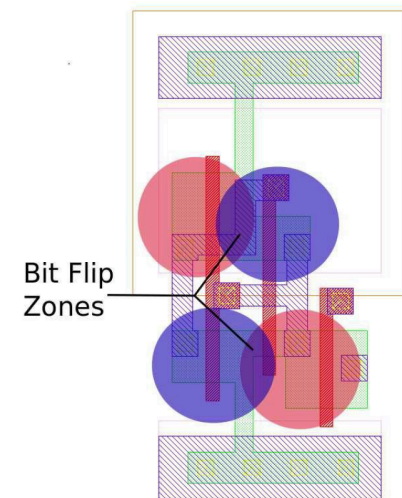


Laser-based Fault Attacks on VLSI (cont.)

- Diameter of a laser beam with respect to layout area affects the fault model
 - Fault type
 - Bit-set [1], bit-reset, bit flip [1]
 - Involved area/spot diameter
 - $\sim 125\mu\text{m} * 125\mu\text{m}$ [1]
 - $0 \sim 2500\mu\text{m}$ [2]
 - Size of affected vector:
 - single bit [1], byte [1, 2]



Laser station [2]

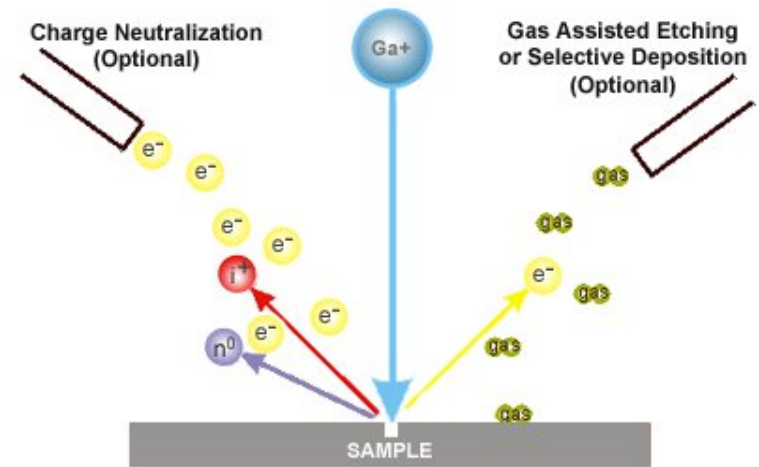


[1] C. Roscian, HOST'13.

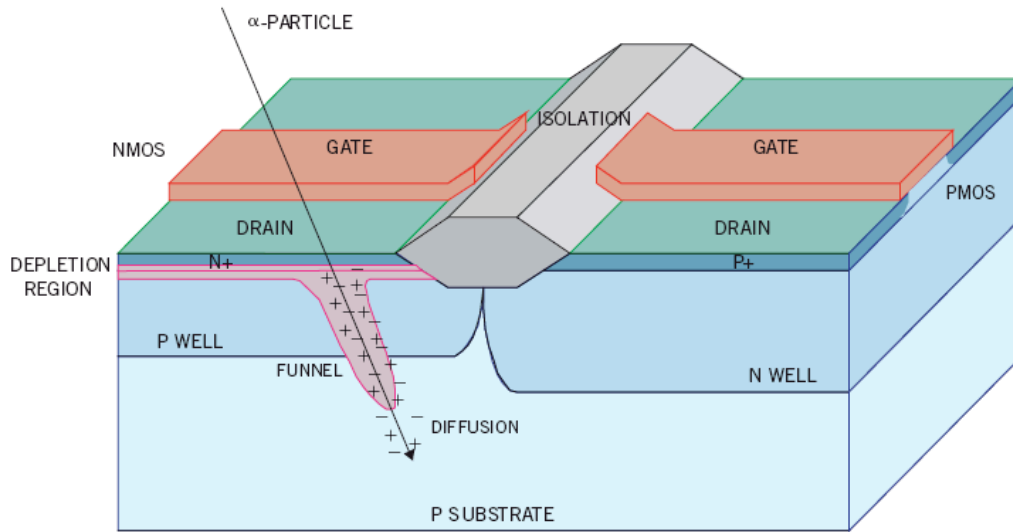
[2] J.-L. Danger and S. Guilley, Fault Attacks on Electronic Circuits, https://perso.telecom-paristech.fr/danger/SETI/DFA_SETI.pdf

Focused Ion Beam based Fault Attacks on VLSI

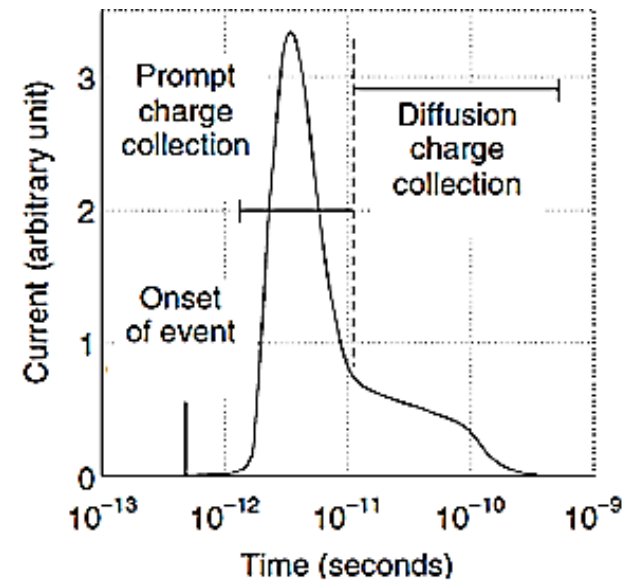
- Focused Ion Beam (FIB)
 - Liquid metal ion sources (gallium)
 - High beam currents can sputter materials at a specific site
 - Cut unwanted electrical connections
 - Deposit conductive material to make a connection



Particle Strike on Chip



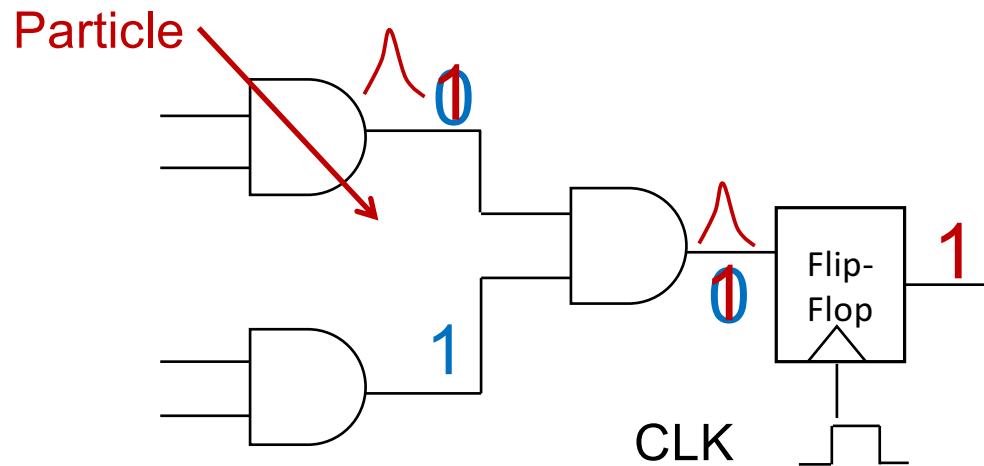
<http://www.iroctech.com/soft-error-library/faq/>



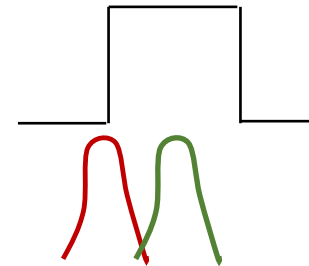
R. Baumann, IEEE Design & Test of Computers'05.

Single Event Transient (SET) Fault

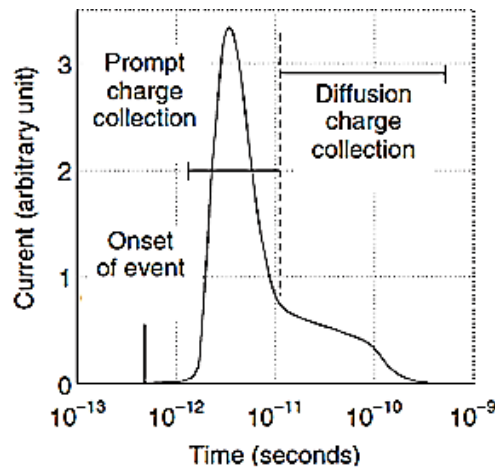
- Factors to consider in fault attacks at hardware level
 - Particle energy
 - Transistor size
 - Fault injection location and timing
 - Masking effects



If the SET is latched, a transient fault is formed

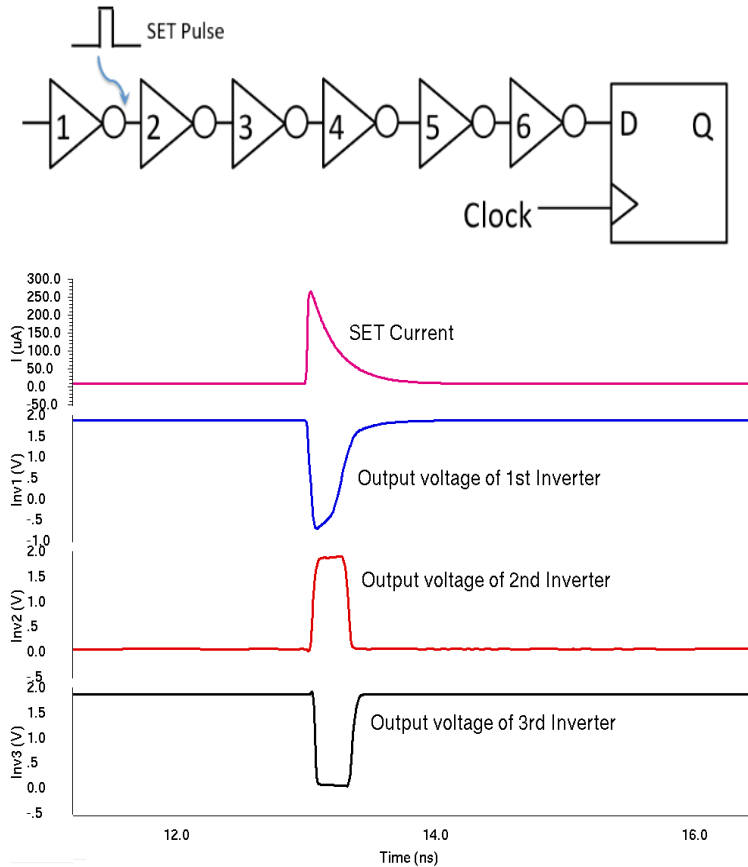


Modeling Single Event Pulse



[1]

$$I(t) = \frac{Q_{coll}}{t_f - t_r} \left(\exp\left(-\frac{t}{t_f}\right) - \exp\left(-\frac{t}{t_r}\right) \right) \quad [2]$$



[3]

[1] R. Baumann, IEEE Design & Test of Computers'05.

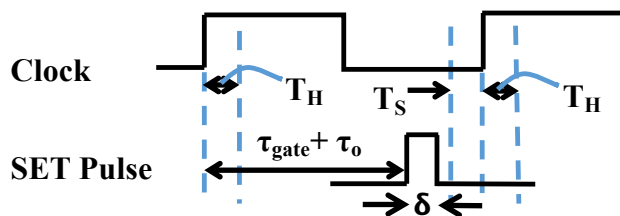
[2] N. Kehl, T-RL'11.

[3] H. Pahlevanzadeh, JETTA'14.

Masking Effect of SETs



- Latch Window Masking



Feature Size ↓

Frequency ↑

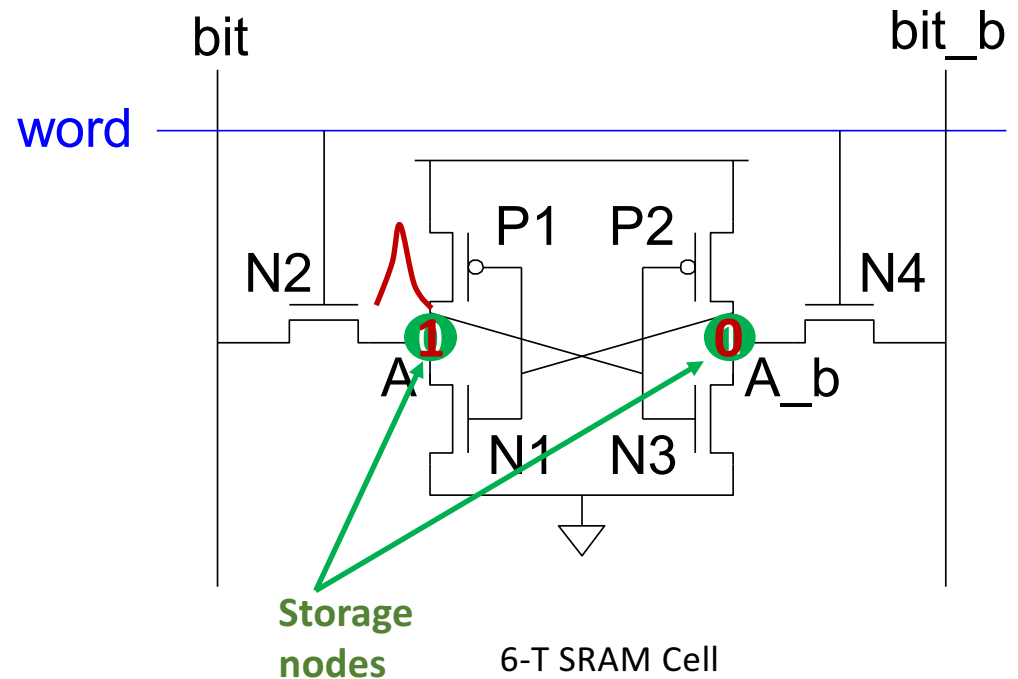
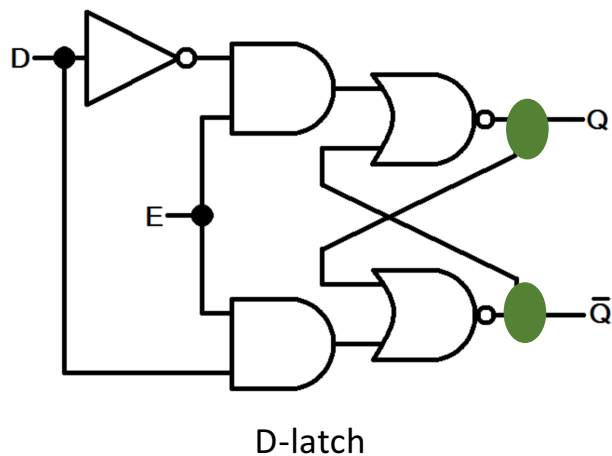
SER_{SET} ↑

Electrical Masking Effect

Latch Window Masking Effect

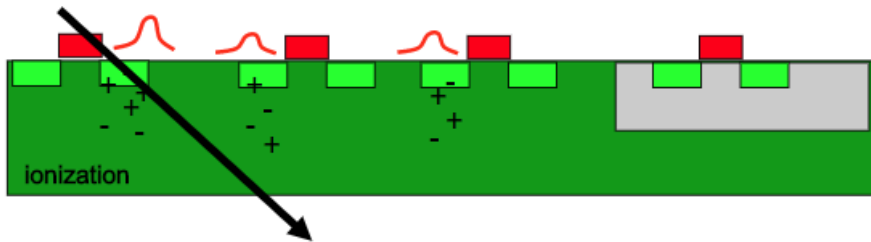
Single Event Upset (SEU) Fault

- SEU permanently changes the logic value saved on the storage element until it is overwritten

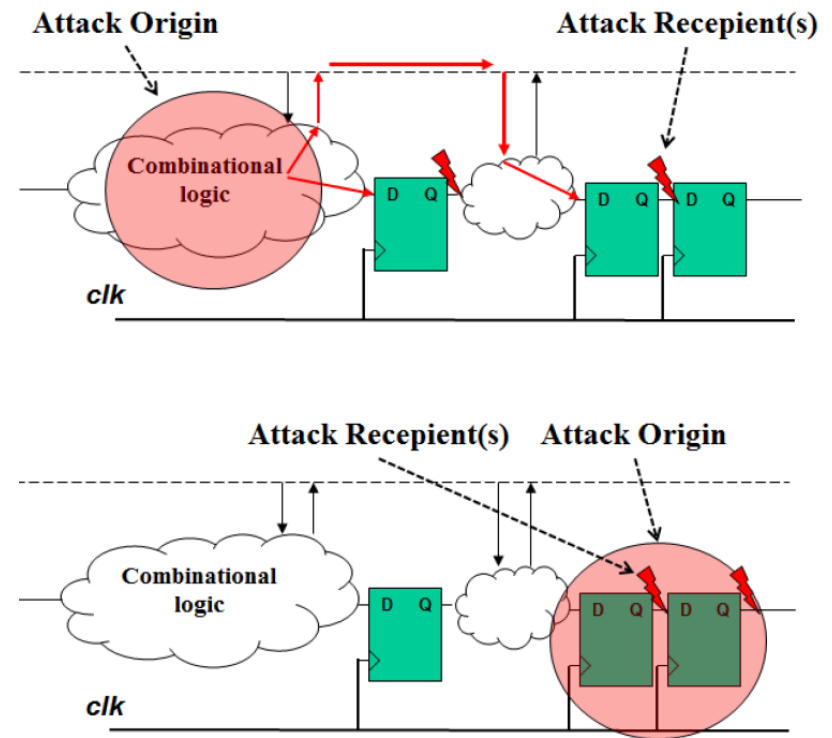


Multiple-bit Upset Faults

- Multiple-bit SEU causes
 - Particle incidence angle
 - Transistor dimensions
 - Voltage supply
 - Memory array density
- Single ion can hit two or more bits causing multiple faults

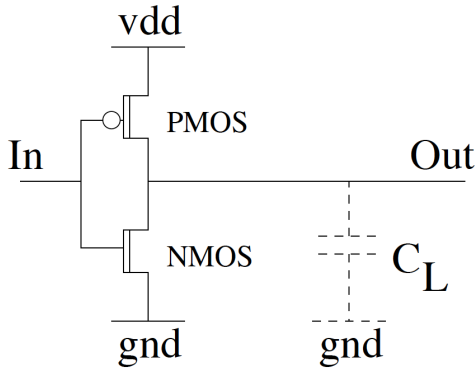
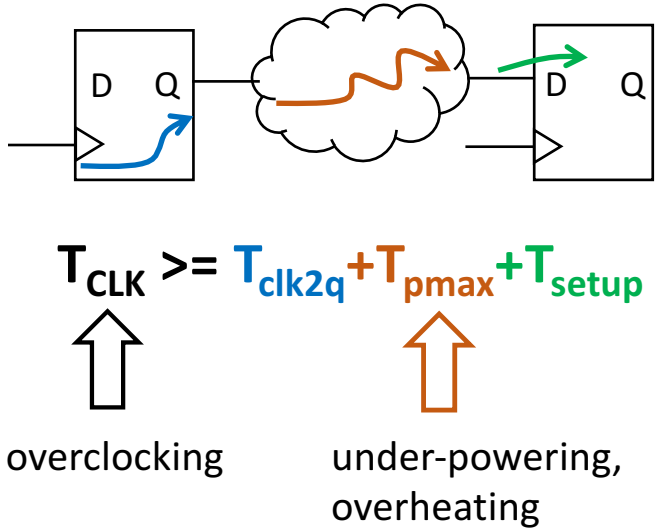


A. Papadimitriou, HAL'16



Overclocking, Under-powering, and Overheating based Fault Attacks on VLSI

- Under-power and overheating fault attacks on circuits lead to the occurrence of timing violation



$$\mu(T) = \mu(T_0) \left(\frac{T}{T_0} \right)^\alpha$$

[2]

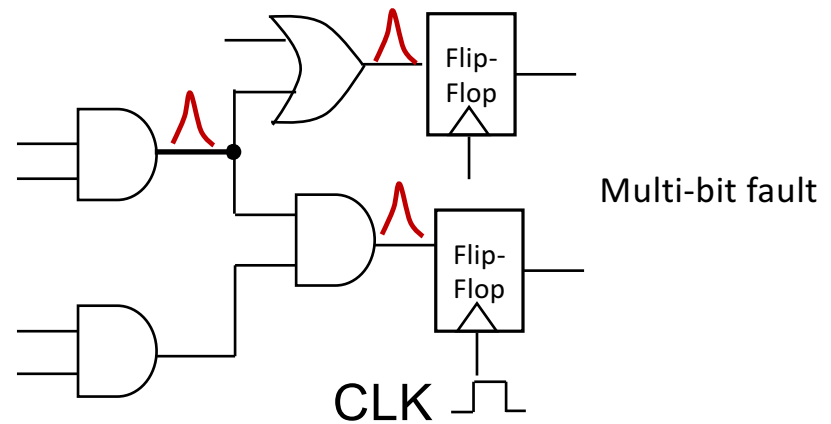
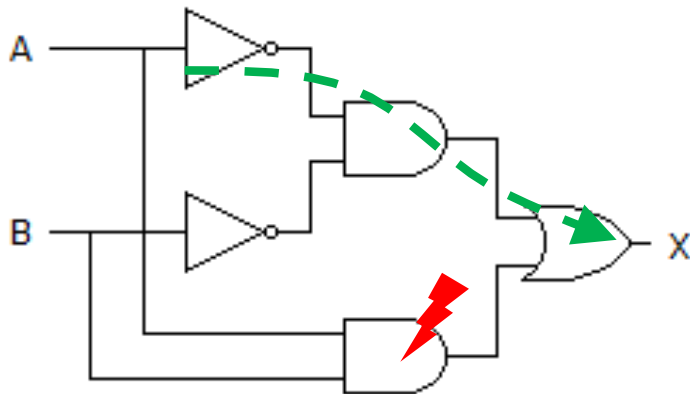
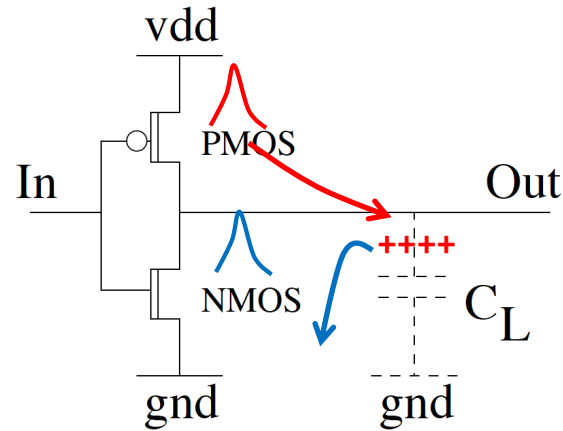
$$t_{pLH} = \frac{C_L \left[\frac{2|V_{th,p}|}{V_{DD} - |V_{th,p}|} + \ln \left(3 - 4 \frac{|V_{th,p}|}{V_{DD}} \right) \right]}{\mu_p C_{ox} \frac{W_p}{L_p} (V_{DD} - |V_{th,p}|)}$$

[1]

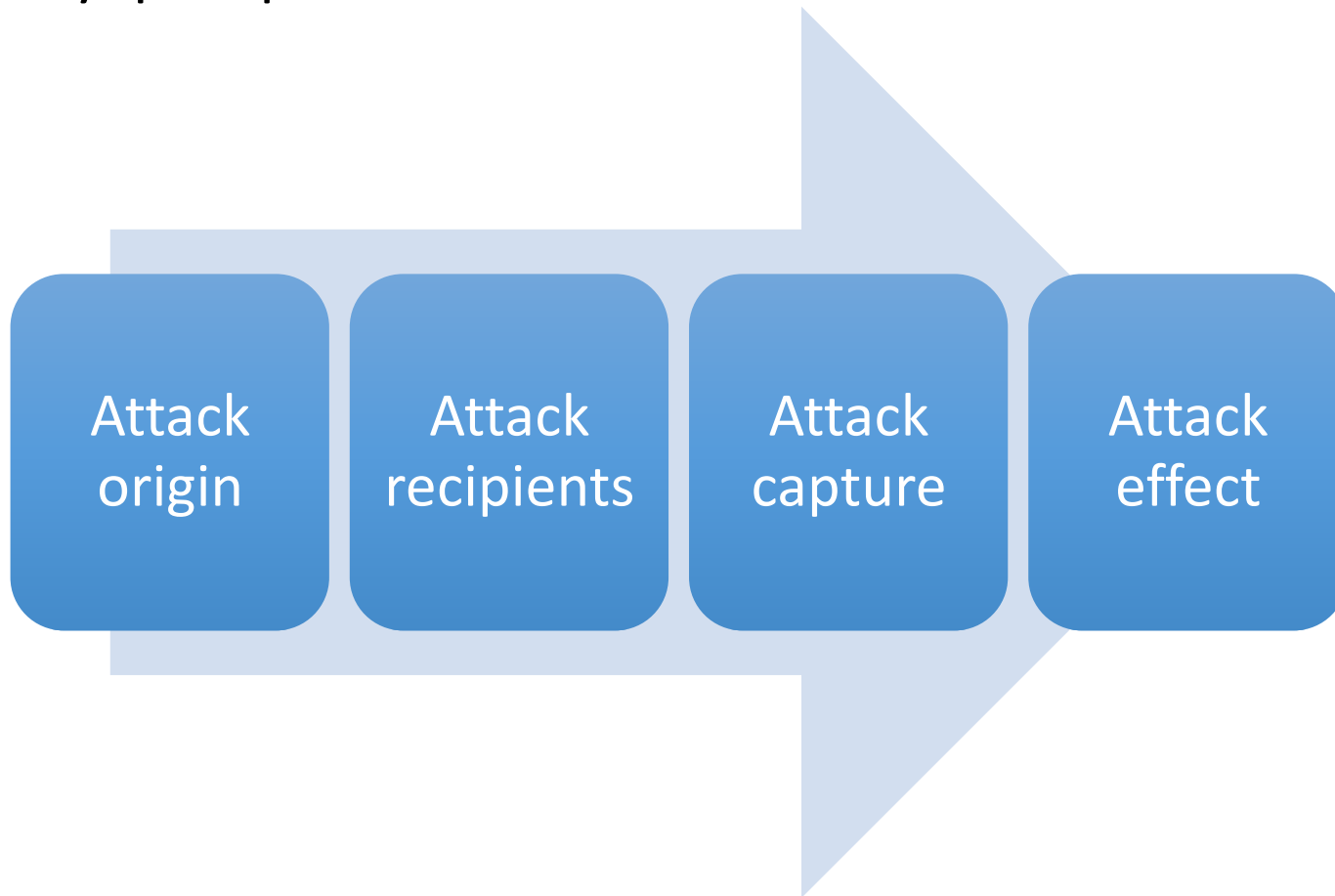
[1] Razavi, Fundamentals of Microelectronics. Wiley, 2008. [2] D. Ha, TLVLSI'12.

Impact of Data Dependence on Fault Injection

- Particle polarity
- Critical path of the design
- Input data



Locality properties of a Fault Model

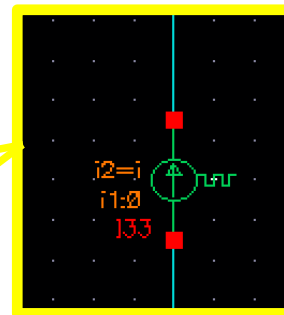
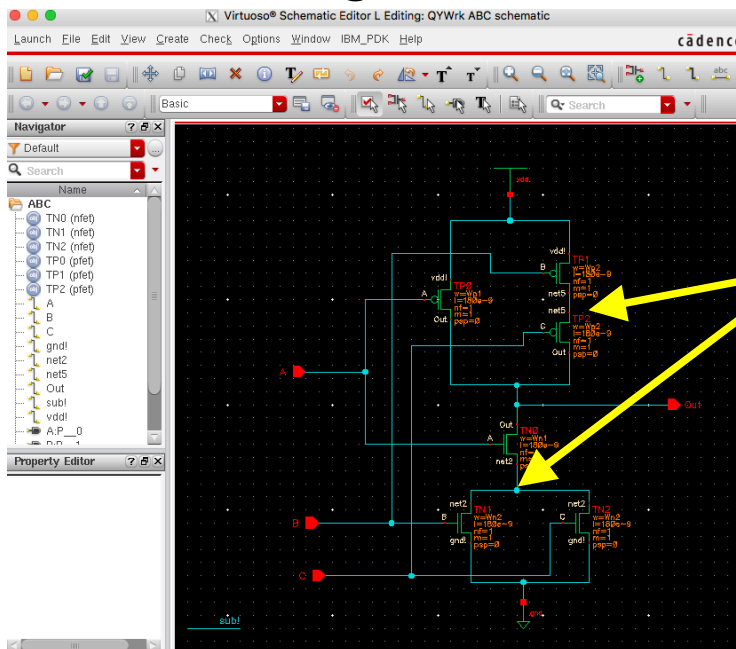


Outline

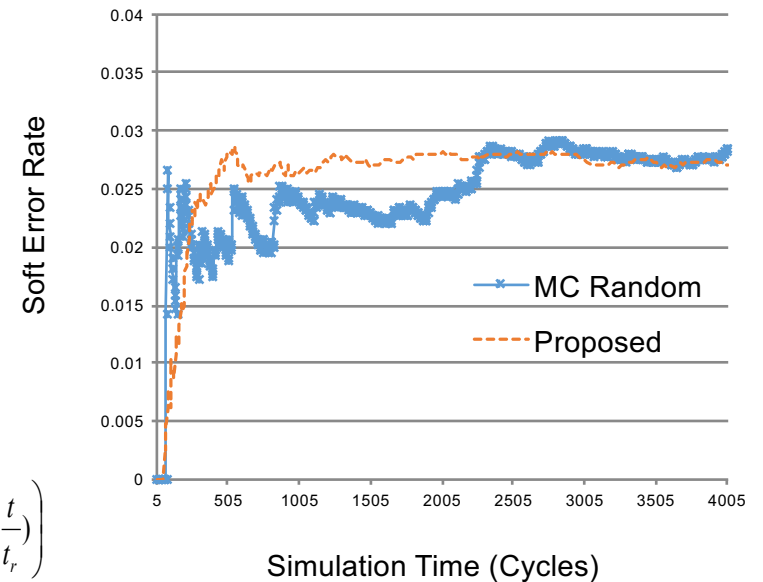
- Introduction of fault model
- Physical mechanisms exploited by fault attacks
- **Fault modeling**
 - Transistor level
 - Gate level
 - Register transfer level (RTL)
- Fault Characterization
 - Overclocking attack
 - Under-powering attack
- Tools for fault injection and analysis

Fault Modeling at Transistor Level

- SPICE-level fault modeling provide accurate results, but it is time-consuming

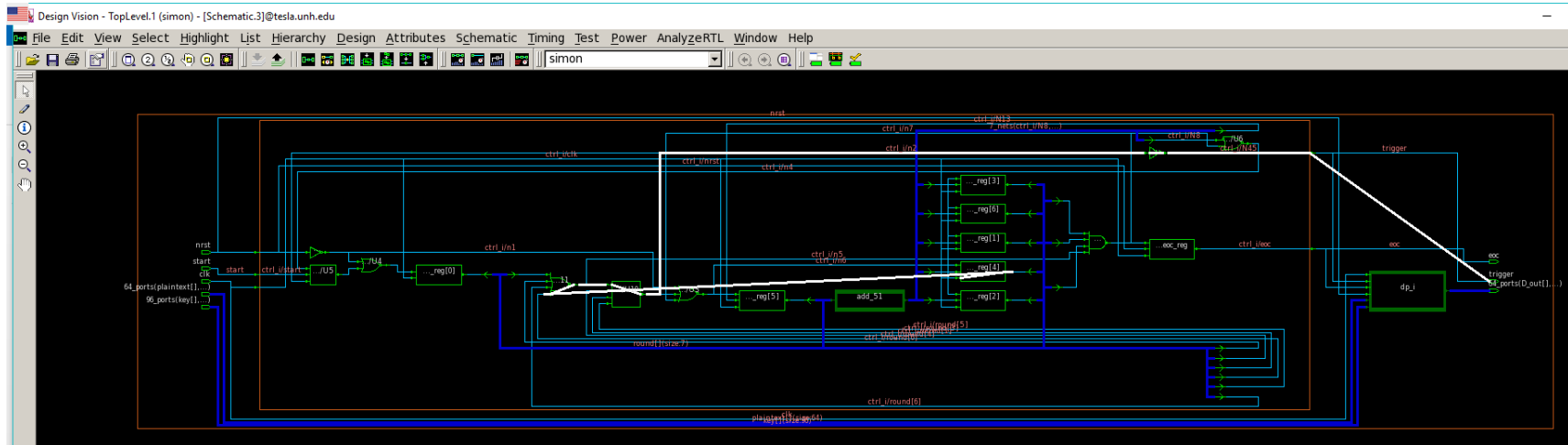


$$I(t) = \frac{Q_{coll}}{t_f - t_r} \left(\exp\left(-\frac{t}{t_f}\right) - \exp\left(-\frac{t}{t_r}\right) \right)$$



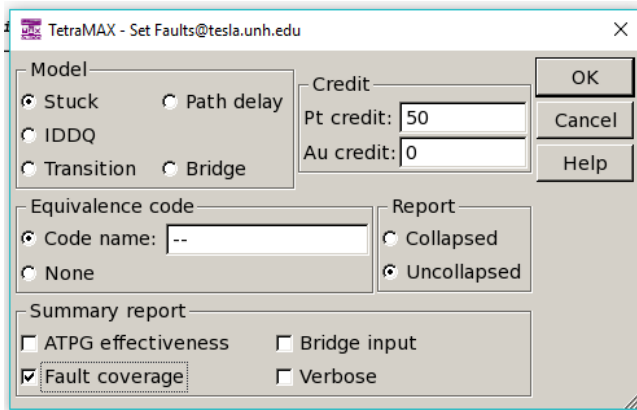
Fault Modeling at Gate Level

- Modules described in hardware description language are first synthesized by Synopsys Design Compiler
- Critical paths are reported by the synthesize tool



Fault Modeling at Gate Level (cont.)

- Simulation stimulus generated by tools, such as TetraMax, are applied to the design module for fault analysis



Fault setup in TetraMAX

Messages	Netlist	Build	DRC	Summary	ATPG	Write Pat.	Write Testbench	Simulation
119	1	45	0/29/20	93.84%	633.31	(10:33.31)		
120	1	44	0/29/20	93.93%	633.34	(10:33.34)		
121	1	43	0/29/20	94.02%	633.36	(10:33.36)		
122	2	41	0/29/20	94.20%	633.38	(10:33.38)		
123	1	40	0/29/20	94.29%	633.38	(10:33.38)		
124	1	39	0/29/20	94.38%	633.40	(10:33.40)		

#patterns stored	#faults detect/active	#ATPG faults red/au/abort	test coverage	process CPU time	
125	1	38	0/29/20	94.46%	633.42 (10:33.42)
126	1	37	0/29/20	94.55%	633.43 (10:33.43)
127	1	36	0/29/20	94.64%	633.44 (10:33.44)
128	1	35	0/29/20	94.73%	633.45 (10:33.45)
129	2	33	0/29/20	94.91%	633.46 (10:33.46)
130	1	32	0/29/20	95.00%	633.47 (10:33.47)
131	1	31	0/29/20	95.09%	633.48 (10:33.48)
132	1	30	0/29/20	95.18%	633.50 (10:33.50)
133	1	29	0/29/20	95.27%	633.52 (10:33.52)
134	1	28	0/29/20	95.36%	633.58 (10:43.58)
135	1	27	0/29/20	95.45%	633.58 (10:43.58)
136	1	26	0/29/20	95.54%	651.65 (10:51.65)
136	0	24	0/31/20	95.54%	651.66 (10:51.66)


```

Uncollapsed Transition Fault Summary Report
-----
fault class      code  #faults
-----
Detected         DT    1070
Possibly detected PT     0
Undetectable     UD     0
ATPG untestable AU     26
Not detected     ND     24

total faults          1120
test coverage         95.54%
fault coverage        95.54%
    
```

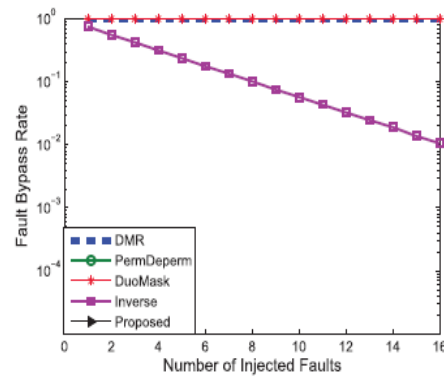
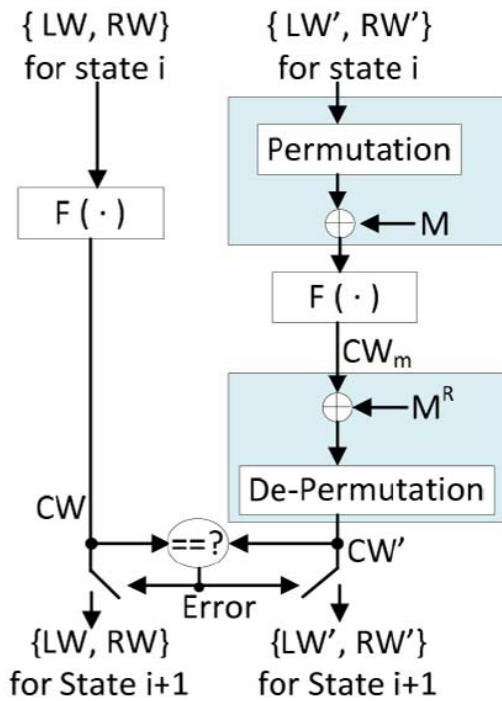


```

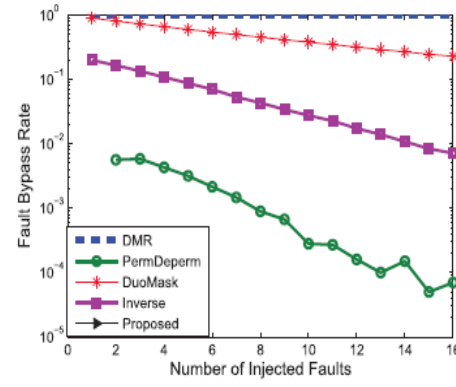
Pattern Summary Report
-----
#internal patterns          136
#full_sequential patterns  136
    
```

Fault coverage report

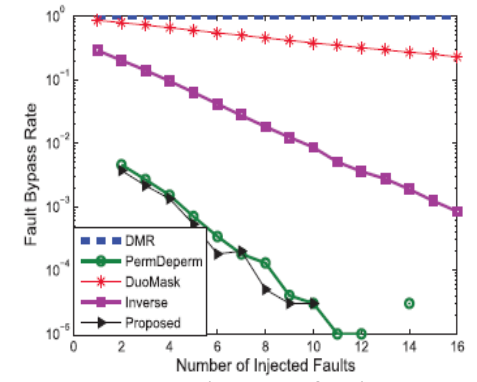
Fault Modeling at Gate Level (cont.)



Bit-flip fault



stuck-at-0 fault

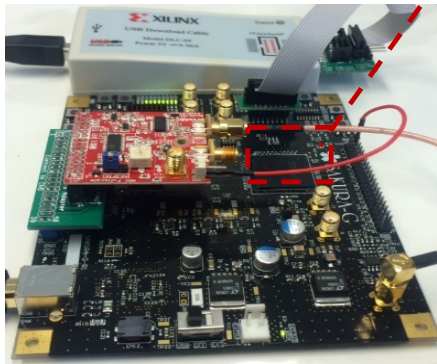


stuck-at-1 fault

Fault-detection algorithm applied to SIMON

Fault Modeling at RTL

- Easy to control the attack origin: area, space or elements



Experimental setup using SAKURA-G FPGA

```
AES_Core aes_core
(.din(dat), .dout(dat_next), .kin(rkey_next), .sel(sel), .FA(rnd[9]));
KeyExpansion keyexpansion
(.kin(rkey), .kout(rkey_next), .rcon(rcon));
```

Fault input
`.FA(rnd[9])`

```
always @(posedge CLK or posedged rst) begin
if (rst) rnd <= 10'b0000_0000_01;
else if (EN) begin
if (Drdy) rnd <= {rnd[8:0], rnd[9]};
else if (~rnd[0]) rnd <= {rnd[8:0], rnd[9]};
end
end

always @(posedge CLK or posedge rst) begin
if (rst) sel <= 0;
else if (EN) sel <= rnd[9];
end

always @(posedge CLK or posedge rst) begin
if (rst) dat <= 128'h0;
else if (EN) begin
if (Drdy) dat <= Din ^ key;
else if (~rnd[0] | sel) dat <= dat_next;
end
end
assign Dout = dat;

always @(posedge CLK or posedge rst) begin
if (rst) key <= 128'h0;
else if (EN)
if (Krdy) key <= Kin;
end

always @(posedge CLK or posedge rst) begin
if (rst) rkey <= 128'h0;
else if (EN) begin
if (Krdy) rkey <= Kin;
else if (rnd[0]) rkey <= key;
else
rkey <= rkey_next;
end
end
```

Consequence : evade the encryption process

Fault in round function of AES

Round function of AES

Fault Modeling at RTL

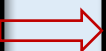
Fault in intermediate register of AES

```
AES_Core aes_core
  (.din(dat), .dout(dat_next), .kin(rkey_next), .sel(sel), .FA(FA));
  KeyExpansion keyexpansion
    (.kin(rkey), .kout(rkey_next), .rcon(rcon));
assign dat_next_fault = dat_next ^ FA;
```

```
always @(posedge CLK or posedge rst) begin
  if (rst)          rnd <= 10'b0000_0000_01;
  else if (EN) begin
    if (Drdy)       rnd <= {rnd[8:0], rnd[9]};
    else if (~rnd[0]) rnd <= {rnd[8:0], rnd[9]};
  end
end

always @(posedge CLK or posedge rst) begin
  if (rst)          sel <= 0;
  else if (EN)      sel <= rnd[9];
end

always @(posedge CLK or posedge rst) begin
  if (rst)          dat <= 128'h0;
  else if (EN) begin
    if (Drdy)       dat <= Din ^ key;
    else if (~rnd[0]|sel) dat <= dat_next;
  end
end
assign Dout = dat;
```



```
always @(posedge CLK or posedge rst) begin
  if (rst)          rnd <= 10'b0000_0000_01;
  else if (EN) begin
    if (Drdy)       rnd <= {rnd[8:0], rnd[9]};
    else if (~rnd[0]) rnd <= {rnd[8:0], rnd[9]};
  end
end

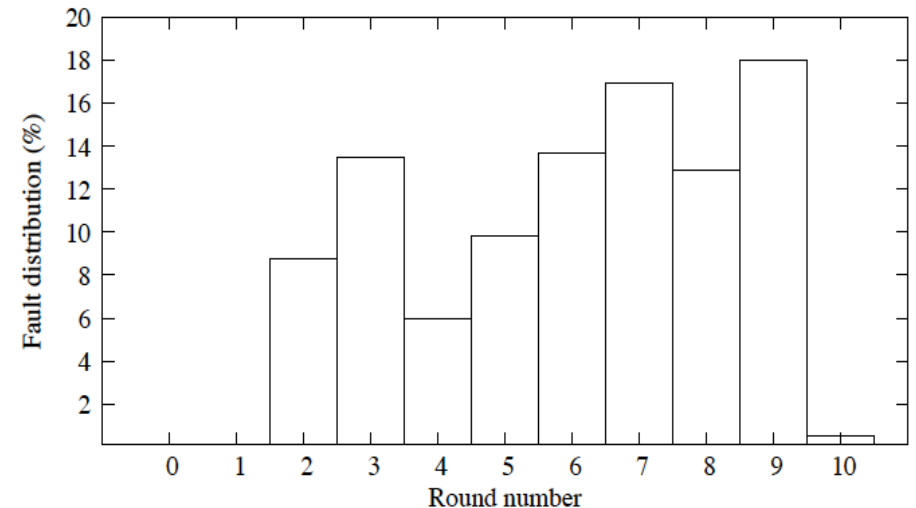
always @(posedge CLK or posedge rst) begin
  if (rst)          sel <= 0;
  else if (EN)      sel <= rnd[9];
end

always @(posedge CLK or posedge rst) begin
  if (rst)          dat <= 128'h0;
  else if (EN) begin
    if (Drdy)       dat <= Din ^ key;
    else if (~rnd[0]|sel) dat <= dat_next_fault;
  end
end
assign Dout = dat;
```

Consequence : Faulty ciphertext

Fault Characterization for Overclocking on FPGA

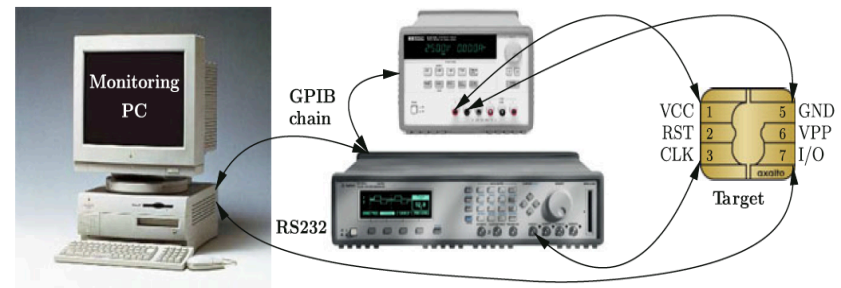
- 1) Send a pair of {plaintext, key} to the test module
- 2) Launch an encryption at nominal settings
 - Obtain a correct ciphertext
- 3) Increase successively by an elementary step the stress applied to the target
 - Obtain a faulty ciphertext
- 4) Process the ciphertext via reversing encryption and compare the intermediate states
- 5) Retrieve the injected fault
 - Collect {plaintext, key, ciphertext, fault}.



Distribution of the faults induced by overclocking attack applied to AES-128 FPGA implementation

Fault Characterization for Under-powering on ASIC

- Standard communication except the power generator
- Record {message, key, ciphertext} for encryptions at different values of VCC
- Key remains at a constant value but input message varies randomly
- Create RTL fault model of AES and inject faults
- Compare simulation output and experimental faulty ciphertext

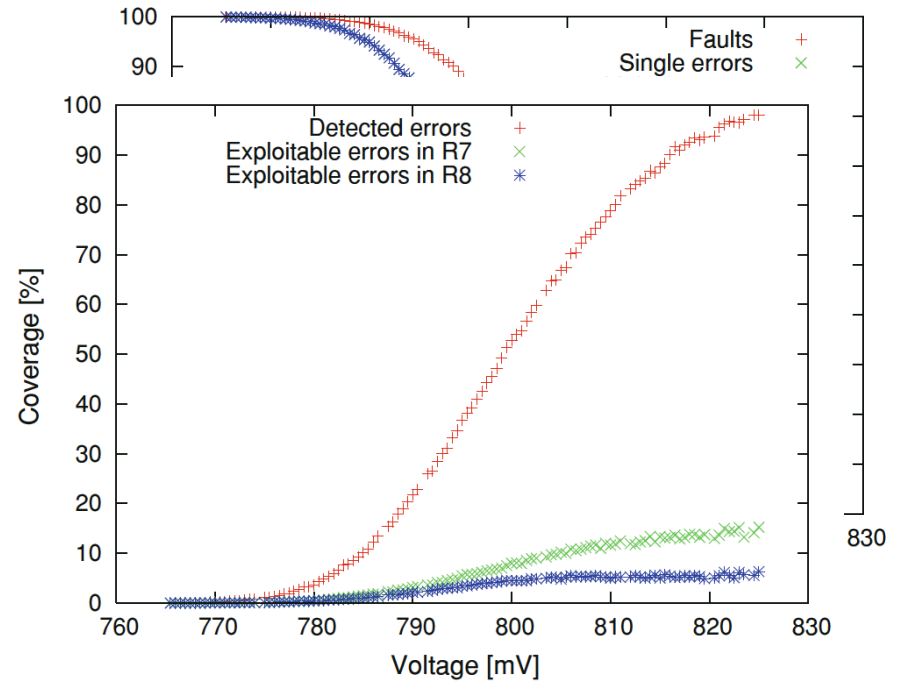
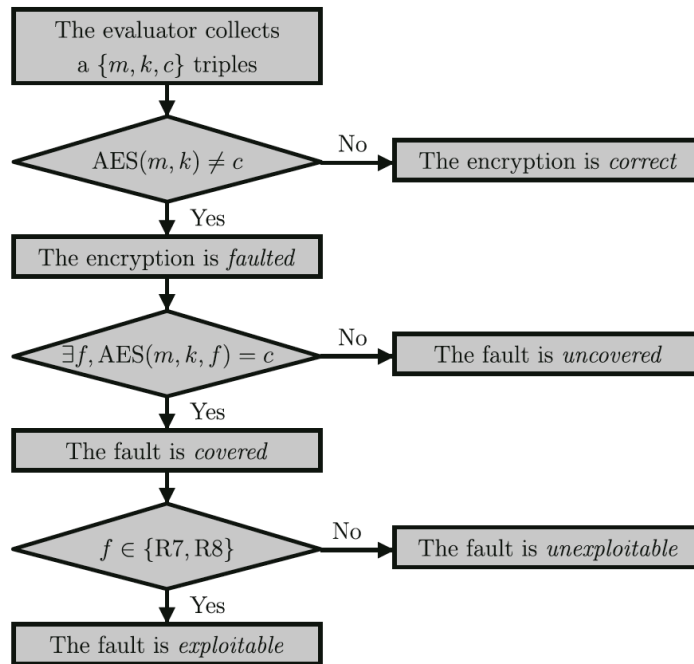


```
for (int round = 1; round < 10; ++round)
{
    // Fault injection
    aes.set_state (aes.get_state () ^ f[round - 1]);
    aes.SubBytes ();
    aes.ShiftRows ();
    aes.MixColumns ();
    aes.AddRoundKey ();
    aes.KeySchedule (round);
}
```

**Bit-flip
fault model**

Fault Analysis on Injected Faults

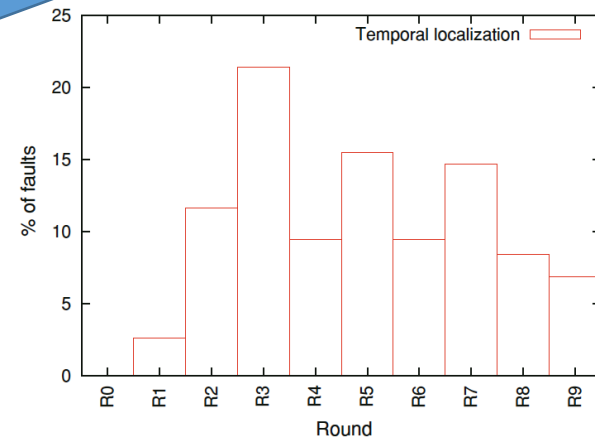
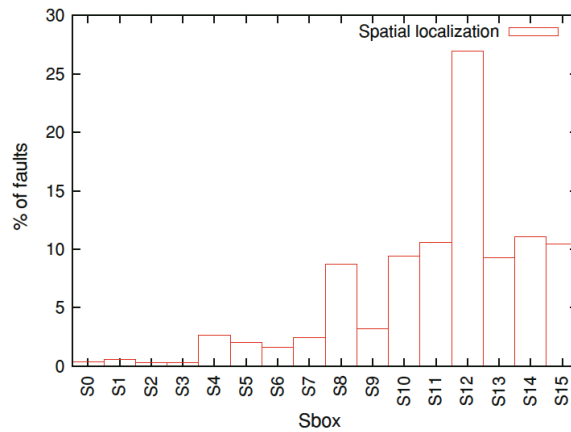
- Flow of fault analysis



Spatial and Temporal Characterization of Faults

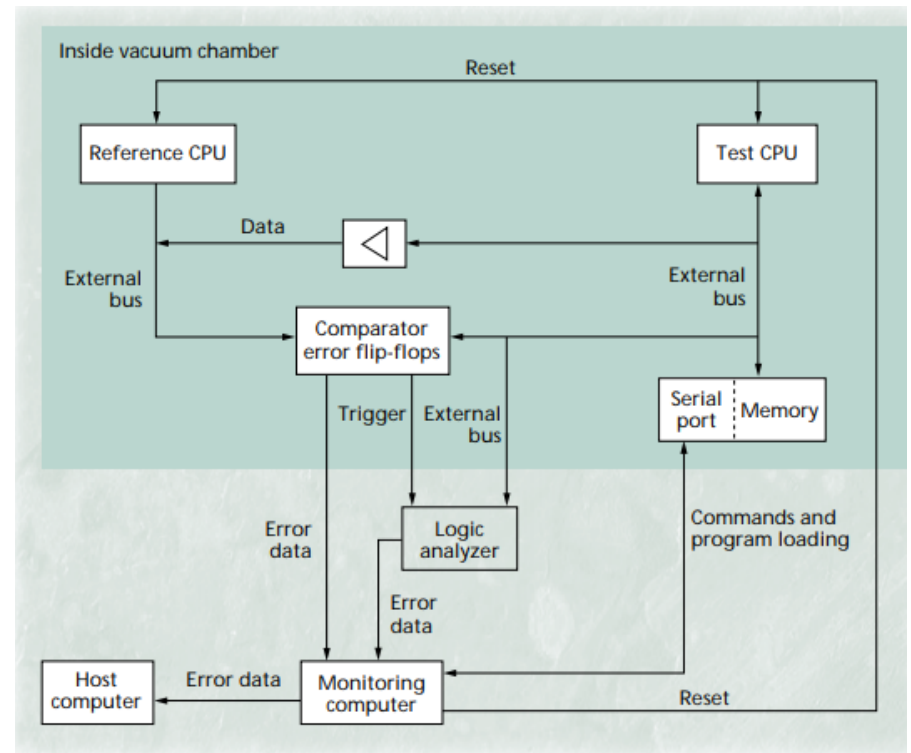
- Spatial characterization
 - In which round
- Temporal characterization
 - In which byte of the state

- Not all S-boxes are placed and routed exactly same
- Faults are localized due to the critical path is data-dependent



Tool for Hardware Fault Injection - FIST

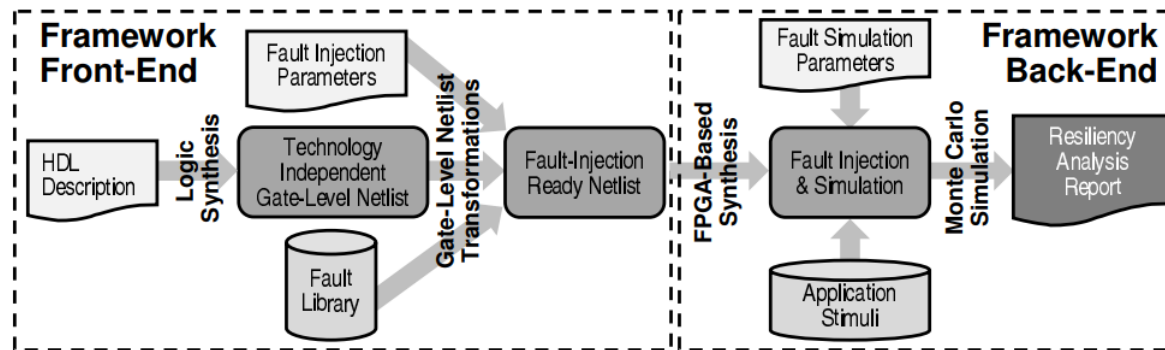
- FIST (Fault Injection System for Study of Transient Fault Effect) can inject faults inside a chip
- This tool uses heavy-ion radiation to generate transient faults at random locations
- The radiation can cause single or multiple-bit errors inside the chip



Platform for Gate level Fault Injection - CrashTest

- This platform

- Is an FPGA-based hardware emulation to performs gate-level fault injection on a full-system design
- Converts the user-provided high-level HDL module to technology independent gate level netlist
- Allows user to specify the fault injection locations as well as random selection of fault sites
- Provides easy way to add new fault models



Bibliophagy

- [C. Giraud, LNCS'04] Giraud C, DFA on AES. In: Dobbertin H., Rijmen V., Sowa A. (eds) Advanced Encryption Standard – AES. AES 2004. Lecture Notes in Computer Science, vol 3373, pp. 27-41. Springer, Berlin, Heidelberg
- [J. Blömer, LNCS'03] Blömer J., Seifert JP. (2003) Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In: Wright R.N. (eds) Financial Cryptography. FC 2003. Lecture Notes in Computer Science, vol 2742, pp. 162-181. Springer, Berlin, Heidelberg
- [P. Dusart, LNCS'03] Dusart P., Letourneux G., Vivolo O. (2003) Differential Fault Analysis on A.E.S. In: Zhou J., Yung M., Han Y. (eds) Applied Cryptography and Network Security. ACNS 2003. Lecture Notes in Computer Science, vol 2846, pp. 293-306. Springer, Berlin, Heidelberg
- [G. Piret, CHES'03] Piret G., Quisquater JJ. (2003) A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad. In: Walter C.D., Koç Ç.K., Paar C. (eds) Cryptographic Hardware and Embedded Systems - CHES 2003. Lecture Notes in Computer Science, vol 2779, pp. 77-88. Springer, Berlin, Heidelberg
- [M. Agoyan, PASTIS'10] Michel Agoyan, Jean-Max Dutertre Dutertre, Amir-Pasha Mirbaha Mirbaha, David , David Naccache Naccache, Anne-Lise Ribotta Ribotta and Assia Tria, Single-Byte Laser Faults using Large Spots 2nd PAcA Security Trends In embedded Security workshop (PASTIS'2010), Gardanne FRANCE 16-17 June 2010. online https://www.emse.fr/~dutertre/doc_recherche/C_2010_1_PASTIS2010.pdf
- [C. Roscian, HOST'13] C. Roscian, J. M. Dutertre and A. Tria, "Frontside laser fault injection on cryptosystems - Application to the AES' last round -," *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Austin, TX, 2013, pp. 119-124
- [R. Baumann, IEEE DES TEST COMPUT'05] R. Baumann, "Soft errors in advanced computer systems," , *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258-266, 2005
- [N. Kehl, T-RL'11] N. Kehl and W. Rosenstiel, "An Efficient SER Estimation Method for Combinational Circuits," in *IEEE Transactions on Reliability*, vol. 60, no. 4, pp. 742-747, Dec. 2011
- [A. Papadimitriou, HAL'16] Athanasios Papadimitriou. RTL modeling of laser attacks for early evaluation of secure ICs and countermeasure design. *Micro and nanotechnologies/Microelectronics*. Université Grenoble Alpes, 2016. English. <NNT : 2016GREAT041>. <tel-01366523>

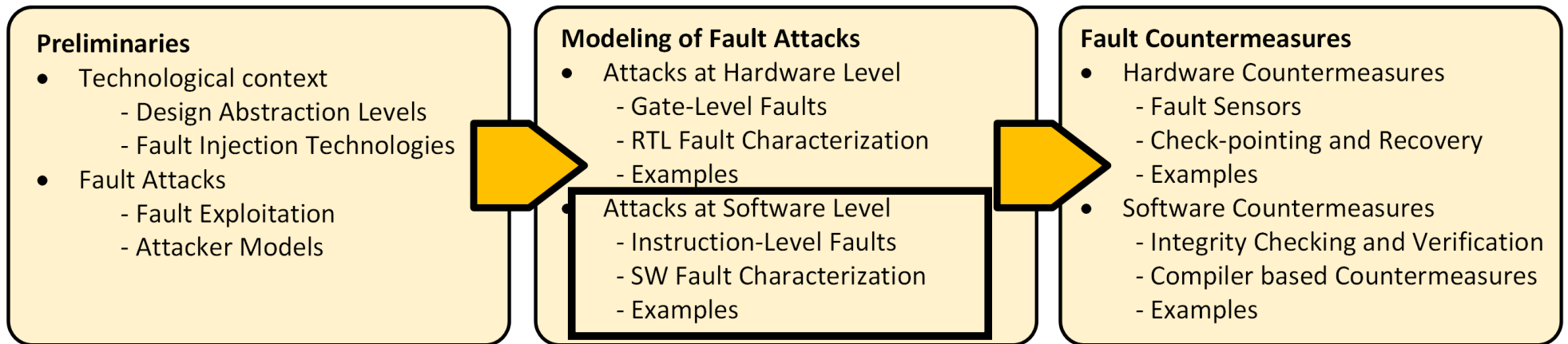
Bibliophagy

- [H. Pahlevanzadeh, JETTA'14] Pahlevanzadeh, H. and Yu, Q, "A New Analytical Model of SET Latching Probability for Circuits Experiencing Single- or Multiple-Cycle Single-Event Transients," *Journal of Electronic Testing*, vol. 30, Issue 5, pp. 595–609, 2014
- [T. Calin, TNS'96] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Trans. Nucl. Sci.*, vol. 43, pt. 1, no. 6, pp. 2874–2878, Dec. 1996.
- [M. D'Alessio, T-DMR'14] M. D'Alessio, M. Ottavi and F. Lombardi, "Design of a Nanometric CMOS Memory Cell for Hardening to a Single Event With a Multiple-Node Upset," in *IEEE Transactions on Device and Materials Reliability*, vol. 14, no. 1, pp. 127-132, March 2014.
- [D. Ha, TLVLSI'12] D. Ha, K. Woo, S. Meninger, T. Xanthopoulos, E. Crain and D. Ham, "Time-Domain CMOS Temperature Sensors With Dual Delay-Locked Loops for Microprocessor Thermal Monitoring," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, pp. 1590-1601, Sept. 2012
- [L. Zussa, DCIS'12] L. Zussa[, et al., "Investigation of timing constraints violation as a fault injection means", *Design of Circuits and Integrated Systems (DCIS)*, 2012
- [S. Guilley, Springer'12] S. Guilley and J.-L. Danger, *Global Faults on Cryptographic Circuits*, Chapter 17 in *Fault Analysis in Cryptography*, Marc Joye and Michael Tunstall (Ed.), Springer, pp. 295-311, 2012
- [A. Papadimitriou, DATE'14] A. Papadimitriou, D. Hély, V. Beroulle, P. Maistri and R. Leveugle, "A multiple fault injection methodology based on cone partitioning towards RTL modeling of laser attacks," *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2014, pp. 1-4.
- [Mei-Chen Hsueh, Computers'97] Mei-Chen Hsueh, T. K. Tsai and R. K. Iyer, "Fault injection techniques and tools," in *Computer*, vol. 30, no. 4, pp. 75-82, Apr 1997.
- [A. Pellegrini, ICCD'08] A. Pellegrini, K. Constantinides, Dan Zhang, S. Sudhakar, V. Bertacco and T. Austin, "CrashTest: A fast high-fidelity FPGA-based resiliency analysis framework," *2008 IEEE International Conference on Computer Design*, Lake Tahoe, CA, 2008, pp. 363-370.

Modeling of Fault Attack at Software Level

Karine Heydemann





Fault attack on software

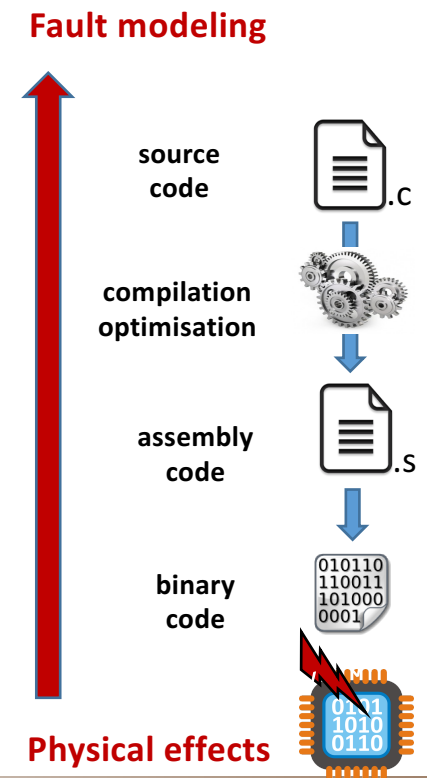
- **Cryptographic key retrieving**
 - By forcing one additional AES round [Dehbaoui et al., COSADE 2013]
 - Different fault exploitations on ARX-like stream ciphers [Kumar et al., FDTC 2017]
- **Bypassing authentication** step in a secure boot process [Timmers et al., FDTC 2016]
- **Taking over a device** by faulting system codes [Timmers et al., FDTC 2017]
- **Privilege escalation** in a TEE-like environment [Vasselle et al. FDTC 2017]
- **Macro view of fault attacks**
 - A successful fault injection leads to an exploitation
 - Useful from an attacker point of view

SW fault characterization

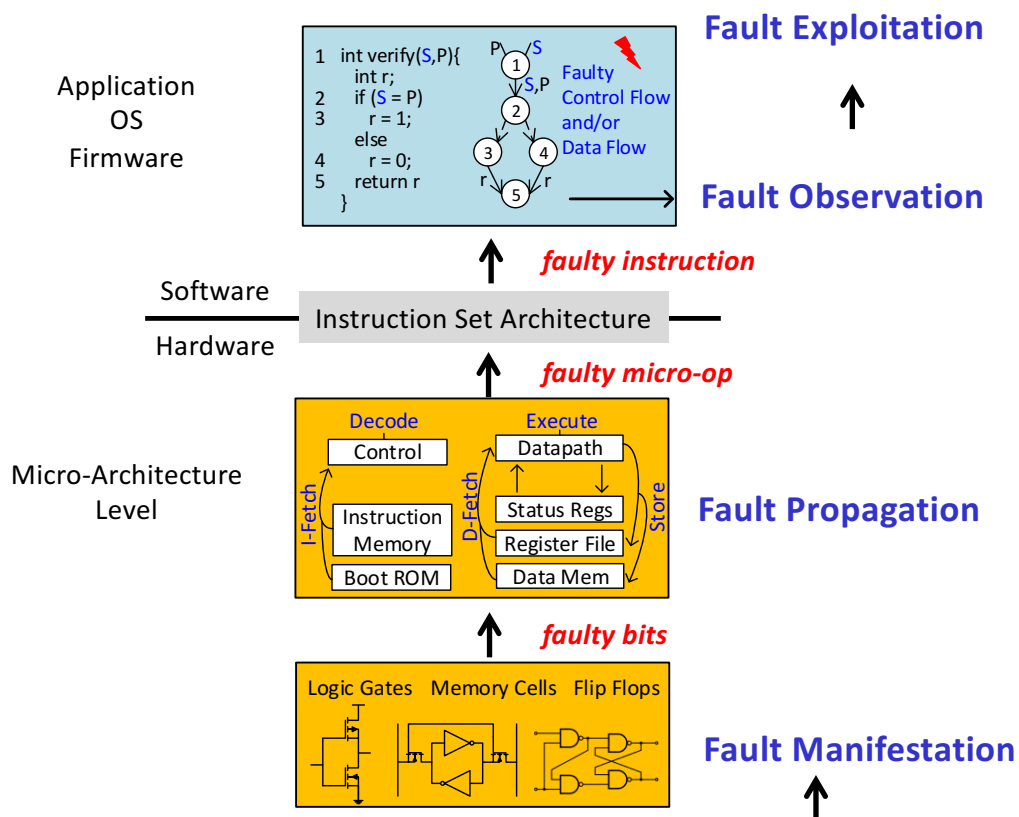
- **Necessary to design** software countermeasures

Fault model

- **Simplified or abstracted representation** of a physical fault effects affecting an embedded software
- At a given code level: binary, assembly code, IR, source code
- No unified methodology to model all possible fault injection impacts on an embedded system (HW + SW)



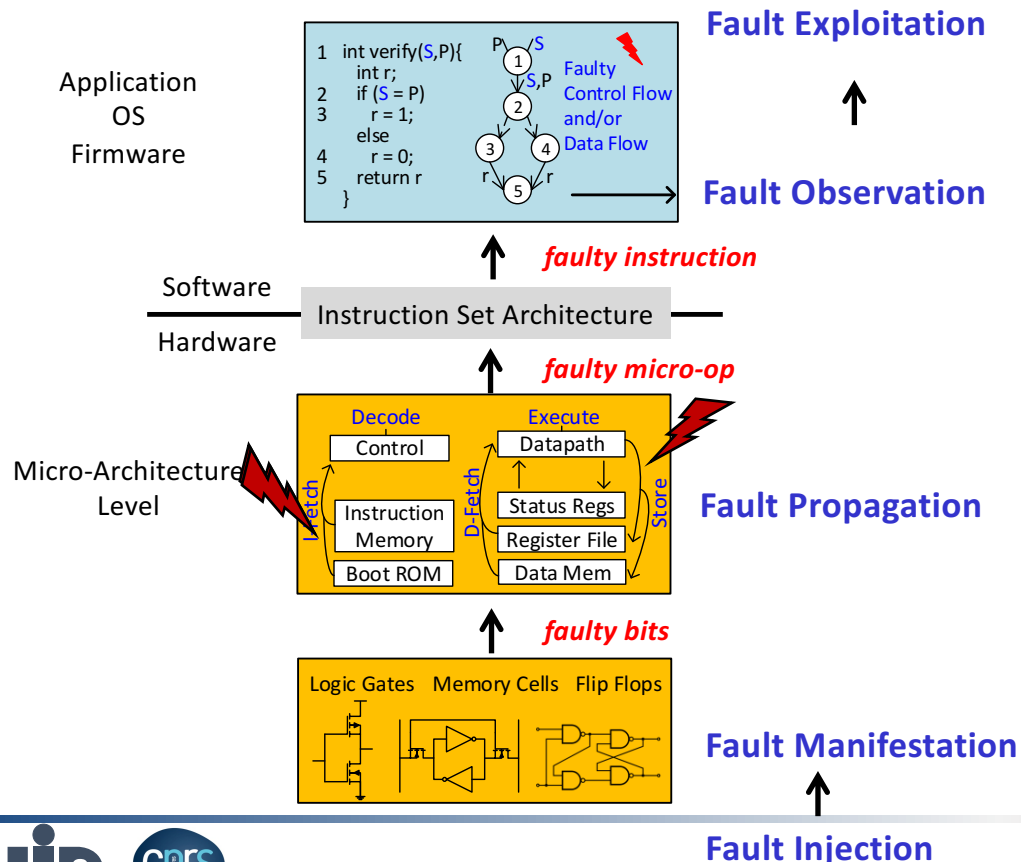
Fault attacks at software level



Fault effect depends on

- Fault injection means
- Running code
- HW target
- Fault location / targeted part of the HW

Fault attacks at software level



Common instruction-level fault models

- Instruction skip
- Instruction replacement
- Register or memory corruption
- Test inversion
- Jump insertion

Application of Attack Potential to Smartcards

– Common Criteria, version 2.9. May 2013

Characterization of faults on the control flow and data flow of software

- **No methodology** or easy way to characterize achievable faults (grey-box model)
- **Huge parameter space** : running code, fault injection mean parameters, HW target
- Common steps for SW fault modeling / characterization:
 1. **Scan the component** to find out areas where faulty outputs are observed
 2. **Select one area** with a high probability to observe a faulty output
 3. **Fault model elaboration** on this selected area

Characterization of faults on the control flow and data flow of software

1. Inject faults while running specific and carefully selected test codes
2. Analyze the output results and infer possible explanations / fault models
3. Validate the fault models
 - By simulation: comparison of observed results with the simulation outputs
 - By refinement: use specifically designed test codes and go back to step 1



Moro et al., *Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller*. FDTC 2013.



Dureuil et al., *From code review to fault injection attacks: Filling the gap using fault model inference*. CARDIS 2015.



Kelly et al., *Characterising a CPU fault attack model via run-time data analysis*. HOST 2017

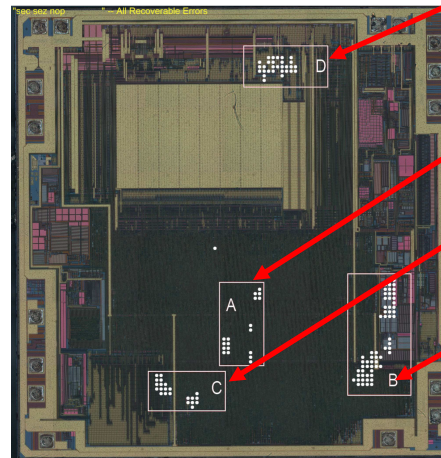
Scan of the HW component

1. Inject faults while running a specific test code w/wo variation of other parameter injection

- Laser injection
- Atmel ATtiny 841 based on AVR core

Fault injections on NOP instructions

4 sensitive areas



Status register fault

Total crash

Memory content corruption

Simultaneous faults on several registers

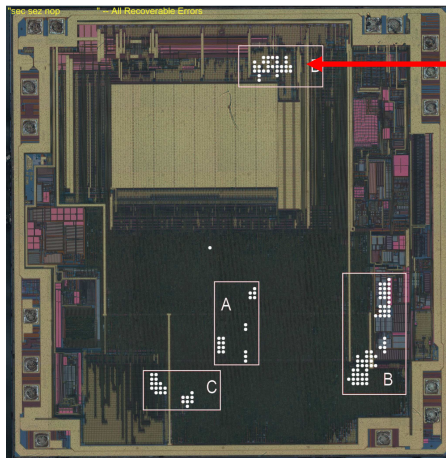


Kelly et al., *Characterising a CPU fault attack model via run-time data analysis*, HOST 2017.

Area selection

2. Selection of the D region

- Laser injection
- Atmel ATtiny 841 based on AVR core



Status register fault

Flags corruption may result in the corruption of a branch instruction

Useful for bypassing security check or secure boot authentication

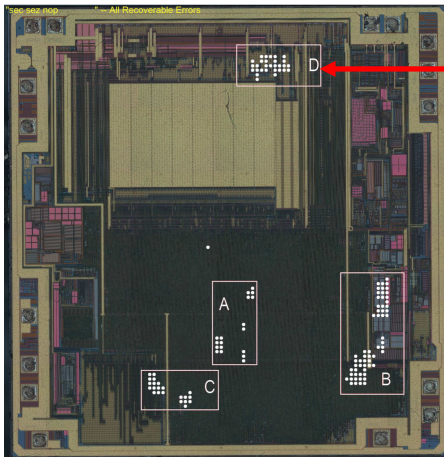


Kelly et al., *Characterising a CPU fault attack model via run-time data analysis*, HOST 2017.

Refinement

3. New fault injections on a specific test code

- Laser injection
- Atmel ATtiny 841 based on AVR core



New test code to test the primarily observed flags vulnerability

- Comparison and branch instructions to different blocks composed of specific MOV instructions
- Easy determination of which instructions have been executed
- Subsequent invalidation of the vulnerability of the flags
- **Final fault model: instruction skip with a high repeatability**



Kelly et al., *Characterising a CPU fault attack model via run-time data analysis*, HOST 2017.

Scan of the HW component

1. Inject faults while running a specific test code w/wo variation of other parameter injection



Moro et al., *Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller*. FDTC 2013.

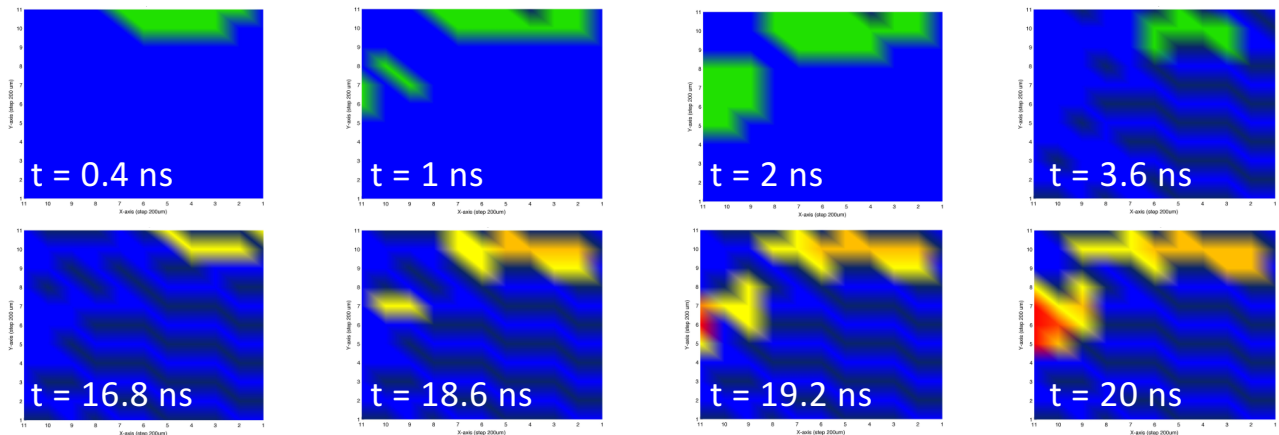
▪ **Target instruction:** single LOAD instruction that loads 0x12345678 into R8

▪ **Spatial and temporal cartography**

▪ **Green** : hardware interrupts

▪ **Red** : faults on the output value

▪ **No fault** on other registers than **R8**
(except for very few faults on R0)

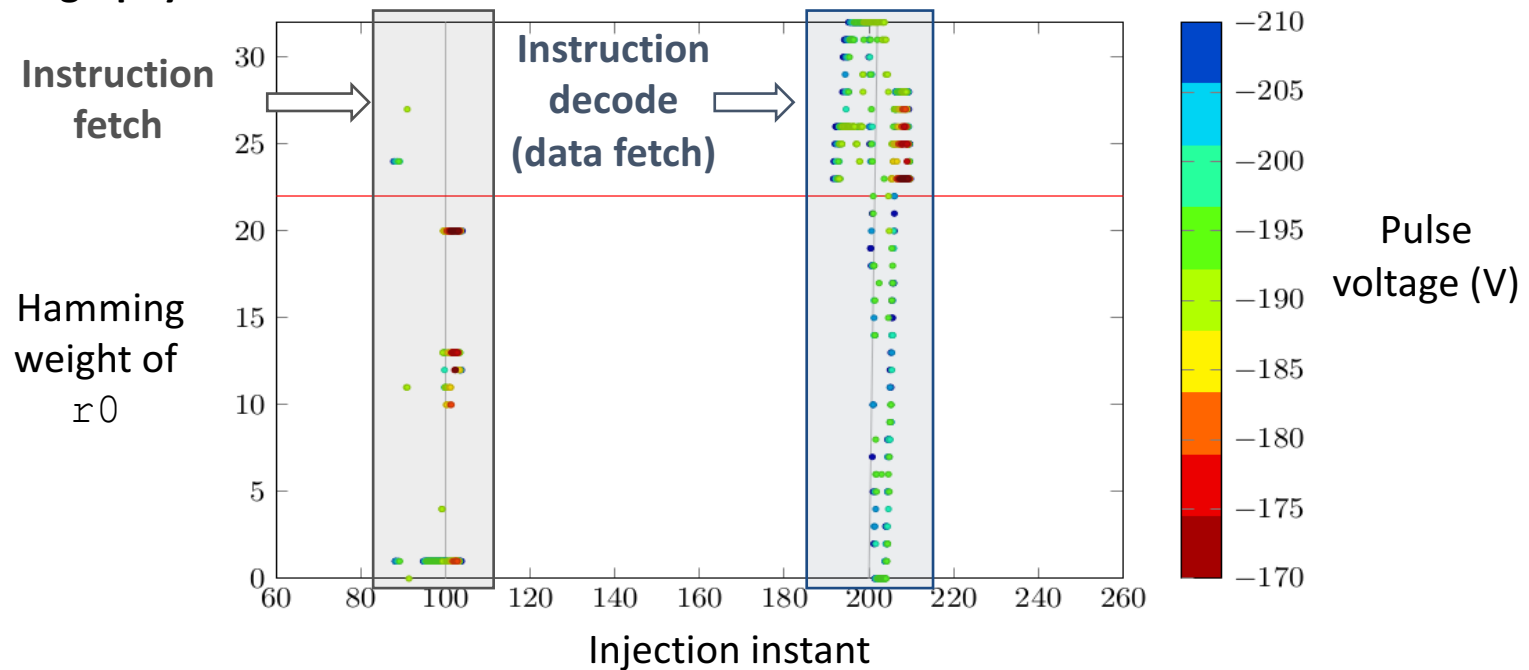


Scan of the HW component

2. Selection of a « working » EM antenna position

▪ **Target instruction:** single LOAD instruction that loads 0x12345678 into R0

▪ **Temporal cartography**



Fault model validation

3. New EM injections on

- Only NOP instructions
- An isolated load instruction surrounded by NOPs

4. Comparison with fault injection simulation

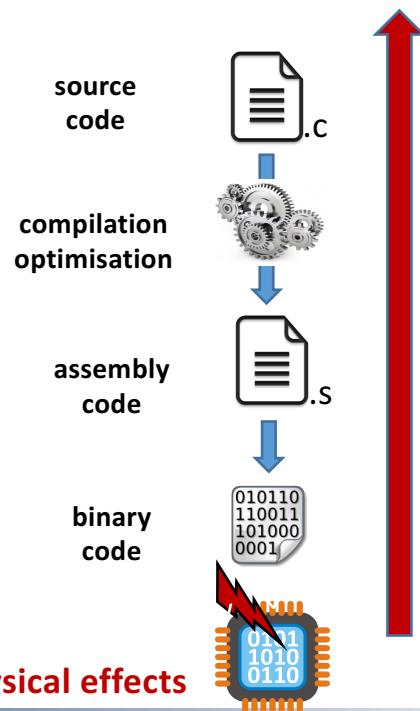
- Instruction replacement fault model
- When no instruction replacement can explain an output → data flow corruption

Validated fault models

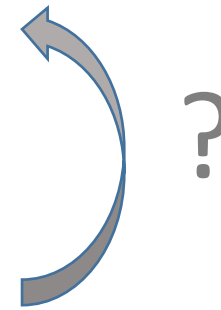
<code>ldr r4, [FLASH_ADDRESS]</code>	→	<code>r4 = attacked_value</code>
<code>inst</code>	→	<code>NOP (25% of observed faults on a larger code)</code>
<code>inst</code>	→	<code>replaced_inst</code>

Fault models at software level

Fault modeling



- At source code level



- At assembly level

- Instruction(s) skip
- Instruction(s) replacement
- Corruption of loaded data
- Register(s) corruption(s)

[Kelly et al., 2017]
[Yuce et al., 2017]
[Timmers et al., 2016]
[Dureuil et al., 2015]
[Rivière et al., 2015]
[Moro et al., 2013]
[Balash et al., 2011]
[Verbauwhede et al., 2011]
[El Bar et al., 2006]

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction writes into a **general register** (add, load, ...)
 - Next use of faulty register will propagate the fault
 - Data corruption

```
a = b + c;
```

```
add    r3, r2, r1  
strb   r3, [r0]
```



```
add    r3, r2, r1  
strb   r3, [r0]
```

```
a = attack();
```

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction writes into a **general register** (add, load, ...)
 - Next use of faulty register will propagate the fault
 - Data corruption

```
a = b + c;
```

```
add    r3, r2, r1  
strb   r3, [r0]
```



```
add    r3, r2, r1  
strb   r3, [r0]
```

```
a = attack();
```

- Equivalent to the corruption of destination register

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction writes into a **general register** (add, load, ...)
 - Next use of faulty register will propagate the fault
 - Potention branch corruption / test inversion

```
cond = *ch;
if (cond)
{
    // do something1
}
else
{
    // do something2
}
```

```
        ldr r3, [r0]
        bnz r3, then
else:   ...
        ...
        j next
then:   ...
        ...
next:   ...
```



```
        ldr r3, [r0]
        bnz r3, then
else:   ...
        ...
        j next
then:   ...
        ...
next:   ...
```

```
cond = *ch;
goto label;then;
if{(!cond)
label_then:
    // do something1
}
else
{
    // do something2
}
```

```
cond = *ch;
goto label_else;
{
    // do something1
}
else
{
label_else:
    // do something2
}
```

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction writes into a **general register** (add, load, ...)
 - Next use of faulty register will propagate the fault
 - Potential branch corruption / test inversion

```
cond = *ch;
if (cond)
{
    // do something1
}
else
{
    // do something2
}
```

```
        ldr r3, [r0]
        bnz r3, then
else:   ...
        ...
        j next
then:   ...
        ...
next:   ...
```



```
        ldr r3, [r0]
        bnz r3, then
else:   ...
        ...
        j next
then:   ...
        ...
next:   ...
```



```
cond = *ch;
goto label_then;
{
label_then:
    // do something1
}
else
{
    // do something2
}
```

```
cond = *ch;
goto label_else;
{
    // do something1
}
else
{
label_else:
    // do something2
}
```

- Equivalent to a transient memory corruption (load instruction)

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction **writes into the status register (flags)**
- Next branch corruption

```
BOOL byteArrayComp(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return BOOL_FALSE;
        }
    }
    return BOOL_TRUE;
}
[Dureuil et al, FISSC Benchmarks, SAFECOMP 2016]
```

- Equivalent to a test inversion or a jump insertion

```
push    {r4, r5, lr}
movs    r3, #0
.L2:
    cmp    r3, r2
    bge    .L7
    ldrb   r5, [r0, r3]
    ldrb   r4, [r1, r3]
    cmp    r5, r4
    bne    .L5
    adds   r3, r3, #1
    b      .L2
.L7:
    movs   r0, #170
    pop    {r4, r5, pc}
.L5:
    movs   r0, #85
    pop    {r4, r5, pc}
```

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction **writes into the status register (flags)**
- Next branch corruption

```
BOOL byteArrayComp(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return BOOL_FALSE;
        }
    }
    return BOOL_TRUE;
}
[Dureuil et al, FISSC Benchmarks, SAFECOMP 2016]
```

- Equivalent to a test inversion or a jump insertion

```
push    {r4, r5, lr}
movs    r3, #0
.L2:
    cmp    r3, r2
    bge    .L7
    ldrb   r5, [r0, r3]
    ldrb   r4, [r1, r3]
    cmp    r5, r4
    bne    .L5
    adds   r3, r3, #1
    b      .L2
.L7:
    movs   r0, #170
    pop    {r4, r5, pc}
.L5:
    movs   r0, #85
    pop    {r4, r5, pc}
```

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction **writes into memory**
 - Data corruption
 - Output corruption and/or propagation of the fault to the subsequent reads
 - **Equivalent to a memory corruption**

```
a = b + c;
```

```
add    r3, r2, r1  
strb   r3, [r0]
```



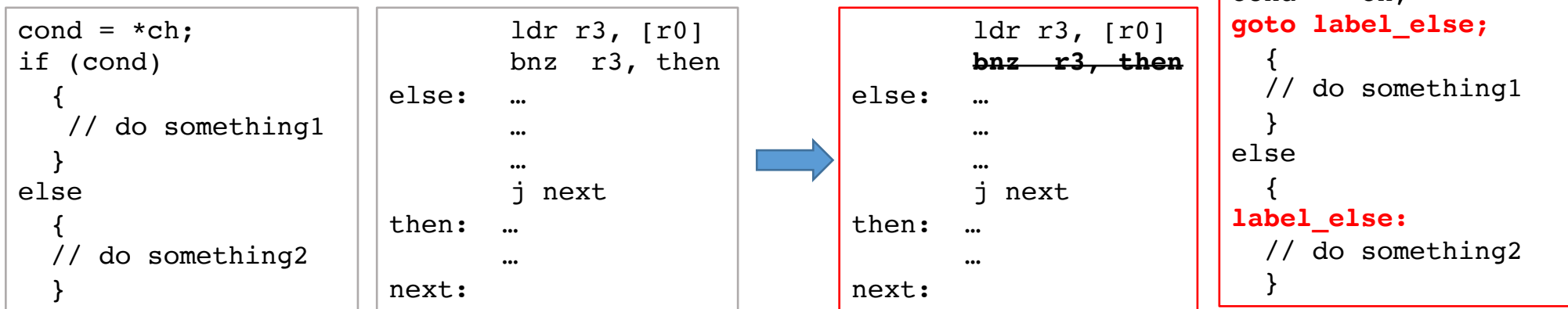
```
add    r3, r2, r1  
strb   r3, [r0]
```

```
a = b + c;  
a = bttack();
```


Fault impact at source level

Instruction skip at assembly level

- The skipped instruction is a **branch**
 - The fall-through block will be executed
 - Potential control-flow corruption

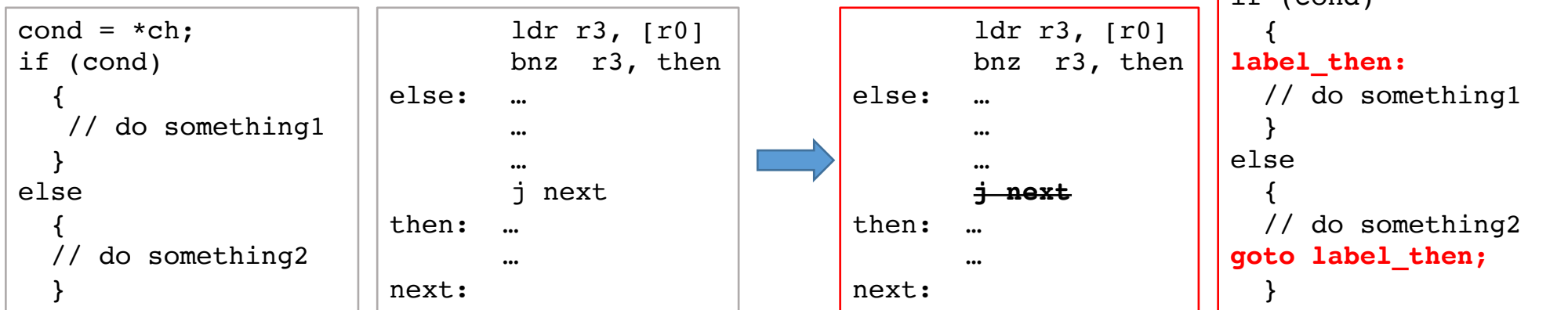


- Equivalent to a jump insertion

Fault impact at source level

Instruction skip at assembly level

- The skipped instruction is a **jump**
 - The fall-through block will be executed
 - Potential control-flow corruption




- Equivalent to a jump insertion

Fault impact at source level

Fault impacting a general register

- Next use(s) of faulty register will propagate the fault
- **One or several consequences**
 - Data corruption(s): **var = attack();**
 - Control-flow corruption: **goto label;**
- Fault propagation related to
 - Subsequent uses of the faulty register: « criticality »
 - Initial code and code compilation/optimization



```
ld  r3, [r0]
st  r3, [r1]
...
bnz r3, then
else:
...
...
j  next
then:
...
next:
```

Fault impact at source level

Instruction replacement

- One instruction is skipped
- One unexpected instruction is executed
- Combination of instruction skip effects with the one of the extra instruction
- From an attacker point of view
 - Only exploitation matters
 - Need to keep the effect controllable:
 - Instruction skip is the most convenient
 - Achievable with different injection means

```
mem_cpy:
    push    {r4, r5, lr}
    movs   r3, #0
.L2:
caps r2, r3, r2
    bge    .L7
    ldrb   r5, [r0, r3]
    strb   r5, [r1, r3]
    bne    .L5
    adds   r3, r3, #1
    b      .L2
.L7:
    movs   r0, #0
    pop    {r4, r5, pc}
```

Fault model at source level

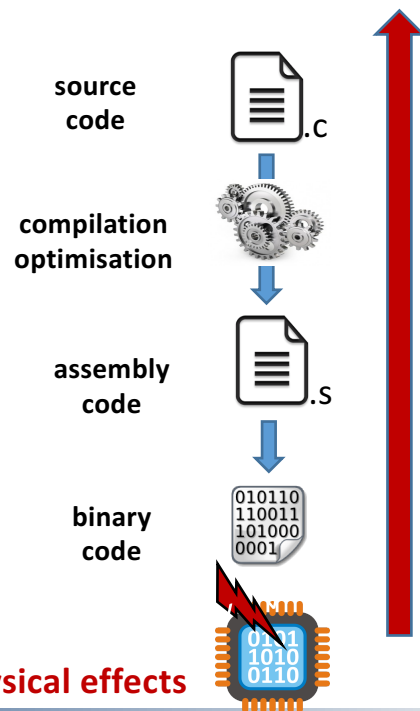
- No correspondence between fault models at instruction-level and source level
 - A statement is translated into several assembly instructions
 - Several faults at assembly level can result into the same fault at source-code level
 - A fault according to a fault model at source code level may not exist once the code is compiled
- Some faults at assembly level cannot be directly expressed at source-code level
 - Code placement, code optimization
- Source-code fault models are necessary
 - Source code protection
 - Vulnerability analysis

Fault models at software level

Fault modeling

[Berthomé et al, 2010]
[Berthomé et al, 2012]

[Kelly et al., 2017]
[Yuce et al., 2017]
[Timmers et al., 2016]
[Dureuil et al., 2015]
[Rivière et al., 2015]
[Moro et al, 2013]
[Balash et al., 2011]
[Verbauwhede et al., 2011]
[El Bar et al., 2006]



- At source code level
 - Control-flow disruption
 - Variable corruption
 - Combination
- At assembly level
 - Instruction(s) skip
 - Instruction(s) replacement
 - Corruption of loaded data
 - Register(s) corruption(s)

References

- [AAPS CC 2013] Common Criteria. (2013). Application of Attack Potential to Smartcards, Version 2.9.
- [Balash et al. 2011] Balasch, J., Gierlichs, B., and Verbauwhede, I. (2011). An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011), pages 105–114. IEEE.
- [Berthomé et al, 2010] Berthomé, P., Heydemann, K., Kauffmann-Tourkestansky, X., and Lalande, J.-F. (2010). Attack model for verification of interval security properties for smart card c codes. In Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS'10)
- [Berthomé et al, 2012] Berthomé, P., Heydemann, K., Kauffmann-Tourkestansky, X., and Lalande, J.-F. (2012). High level model of control flow attacks for smart card functional security. In Proceedings of the 7th International Conference on Availability, Reliability and Security (ARES'12), pages 224–229, IEEE Computer Society.
- [Dehbaoui et al. 2013] Amine Dehbaoui, A., Mirbaha, A-P., Moro, N., Dutertre, J-M., Tria, A.: Electromagnetic Glitch on the AES Round Counter. COSADE_2013: 17-31
- [Dureuil et al. 2015] Dureuil, L., Potet, M., de Choudens, P., Dumas, C., and Clédière, J. (2015). From code review to fault injection attacks: Filling the gap using fault model inference. In Proceedings of the 14th International Conference Smart Card Research and Advanced Applications (CARDIS), volume 9514 of LNCS, Springer.
- [Dureuil et al, FISSC, 2016] Dureuil, L., Petiot, G., Potet, M.-L., Crohen, A., and Choudens, P. D. (2016). FISSC: a Fault Injection and Simulation Secure Collection. In Proceedings of International Conference on Computer Safety, Reliability and Security, volume 9922 of (SAFECOMP), pages 3–11, Trondheim.
- [El Bar et al., 2006] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. (2006). The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE, 94(2):370–382.

References

- [Kelly et al., 2017] M.S.Kelly, K.Mayes, J.F.Walker. Characterising a CPU fault attack model via run-time data analysis. In 2017 IEEE International Symposium on Hard- 1268 ware Oriented Security and Trust (HOST). 79–84.
- [Kumar et al., 2017] S. V. Dilip **Kumar** ; Sikhar Patranabis ; Jakub Breier ; Debdeep Mukhopadhyay ; Shivam Bhasin ; Anupam Chattopadhyay ; Anubhab Baksi. A Practical Fault Attack on ARX-Like Ciphers with a Case Study on ChaCha20. 2017 Workshop Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC), 2017. IEEE.
- [Rivière et al. 2015] Rivière, L., Najm, Z., Rauzy, P., Danger, J.-L., Bringer, J., and Sauvage, L. (2015). High precision fault injections on the instruction cache of armv7-m architectures. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2015), Washington, United States.
- [Timmers et al., 2016] Timmers, N., Spruyt, A., and Witteman, M. (2016). Controlling PC on ARM Using Fault Injection. 2016 Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC 2016), pages 25–35, Santa Barbara, CA, USA. IEEE.
- [Timmers et al., 2017] Timmers, N., Mune, C. Escalating Privileges in Linux Using Voltage Fault Injection. Workshop Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC), 2017. IEEE.
- [Vasselle et al., 2017] Vasselle, A., Thiebauld, H., Maouhoub, Q., Morisset, A. and Ermeneux, S. Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot. 2017 Workshop Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC), 2017. IEEE.
- [Verbauwhede et al., 2011] Verbauwhede, I., Karaklajic, D., and Schmidt, J.-M. (2011). The Fault Attack Jungle - A Classification Model to Guide You. In Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011), IEEE.
- [Yuce et al., 2017] Yuce, B., Ghalaty, N. F., Deshpande, D., Santapure, H., Conor, P., Nashandali, L., and Schaumont, P. (2017) Analyzing the Fault Injection Sensitivity of Secure Embedded Software. ACM Trans. Embedded Comput. Syst. 16(4): 95:1-95:25

A Whitebox Introduction to Fault Attacks

Patrick Schaumont

schaum@vt.edu

Professor

Karine Heydemann

karine.heydemann@lip6.fr

Associate Professor

Qiaoyan Yu

qiaoyan.yu@unh.edu

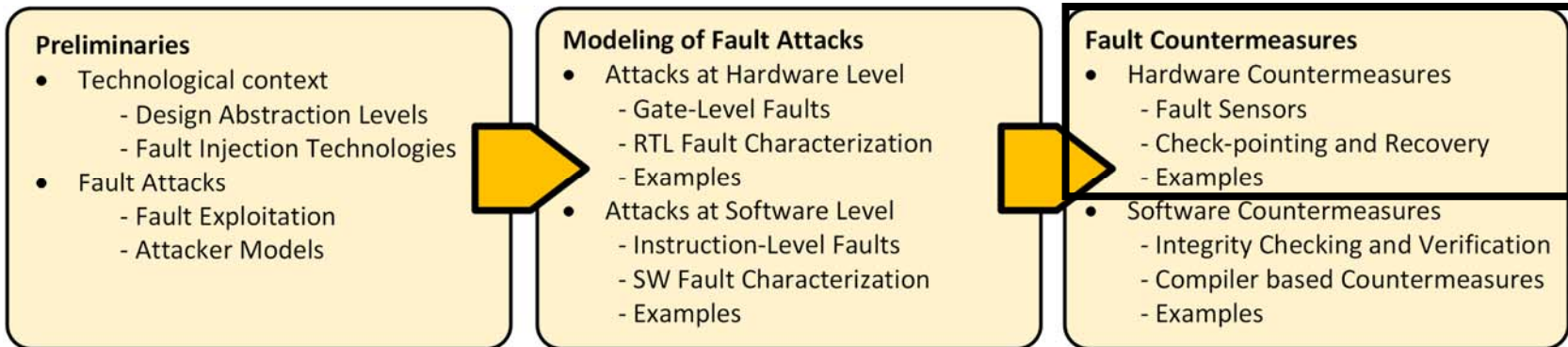
Associate Professor



Part 3: Countermeasures

Hardware Countermeasures

Patrick Schaumont



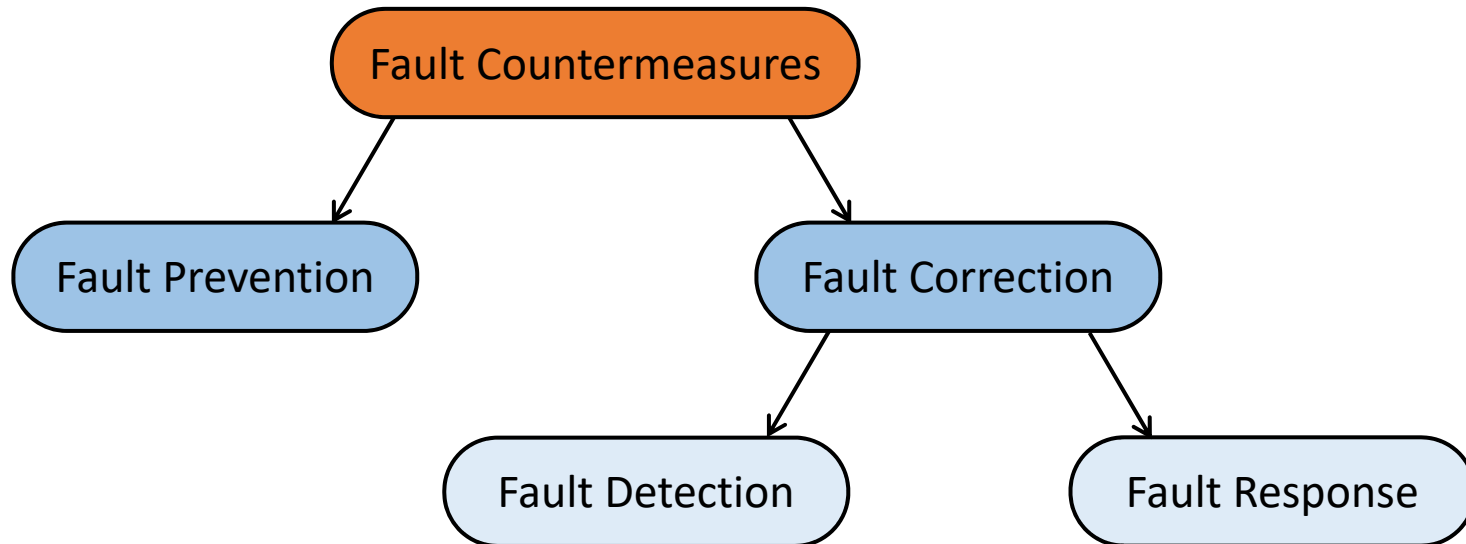
Outline

1. Taxonomy of Countermeasures
2. Fault Prevention
3. Fault Detection
4. Fault Response

Outline

1. Taxonomy of Countermeasures
2. Fault Prevention
3. Fault Detection
4. Fault Response

Taxonomy of Countermeasures



- Physical Shielding
- Filtering
- Logical Shielding
 - Time Jitter
 - Randomization

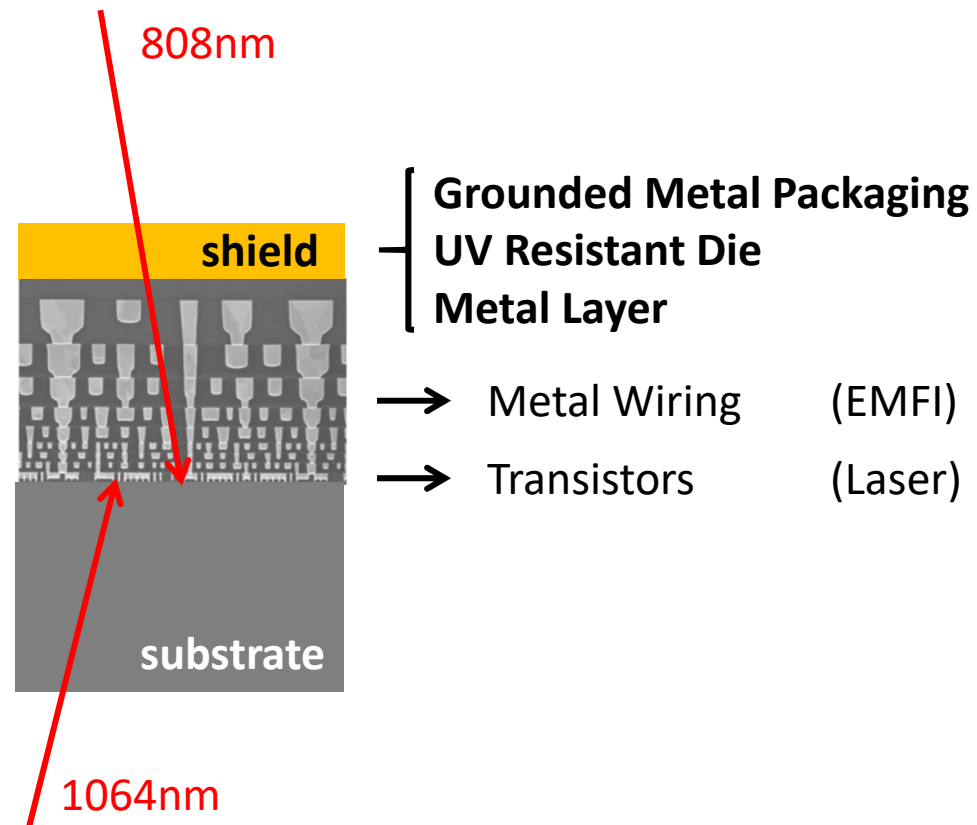
- Sensors
 - In-situ vs Environment
 - Global vs Local
- Concurrred Error Detection
 - Time
 - Spatial
 - Information
 - Algorithm-specific

- Infection
- Checkpoint-Restore
- Redundancy

Outline

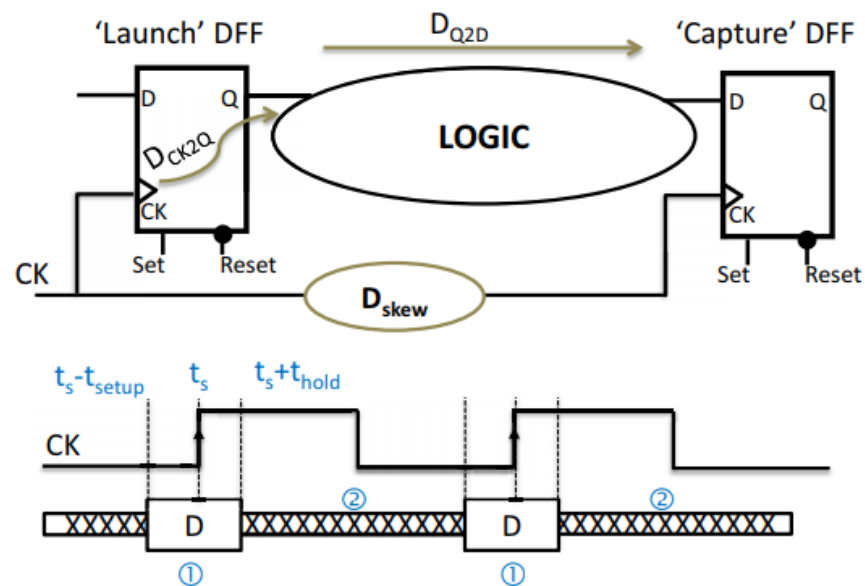
1. Taxonomy of Countermeasures
2. Fault Prevention
3. Fault Detection
4. Fault Response

Shielding and Filtering



Logical Shielding

- Insertion of random delays, clock jitter
- Internal/modulated clocks
- Randomization of schedule

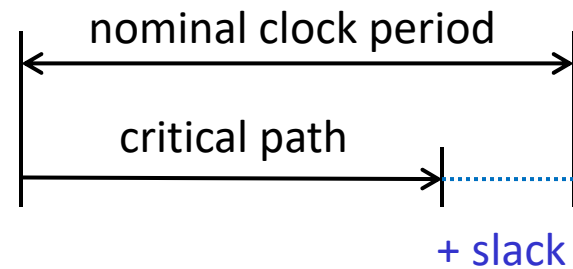
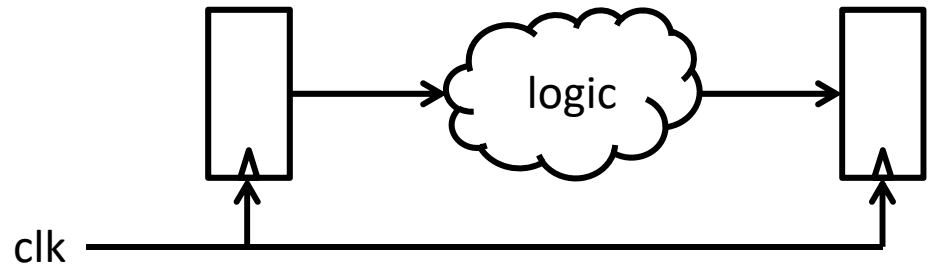


Outline

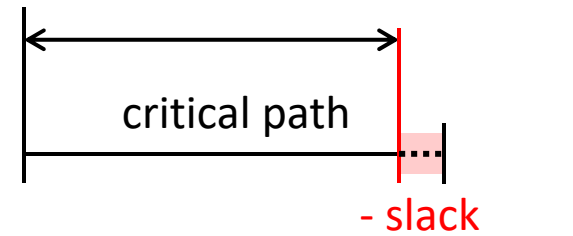
1. Taxonomy of Countermeasures
2. Fault Prevention
3. Fault Detection
4. Fault Response

Sensors

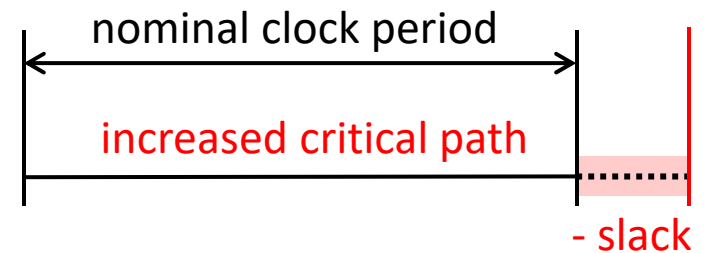
- Parameter
 - Timing
 - Voltage
 - Photon
 - Temperature
 - EMFI



- Overclocking
- Clock Glitching



- Underfeeding
- Voltage Glitching
- Overheating



Sensors

- Parameter
 - Timing
 - Voltage
 - Photon
 - Temperature
 - EMFI
- Locality
 - In-situ – 100% detection rate
 - Local Environment – false-positive/false-negative
 - Global Environment – false-positive/false-negative

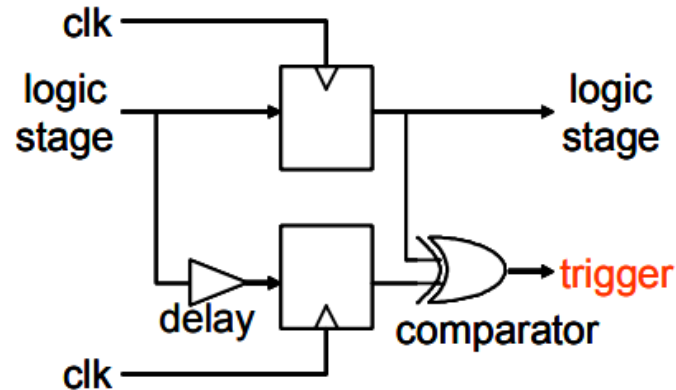
Sensors

- Parameter

- Timing
- Voltage
- Photon
- Temperature
- EMFI

- Locality

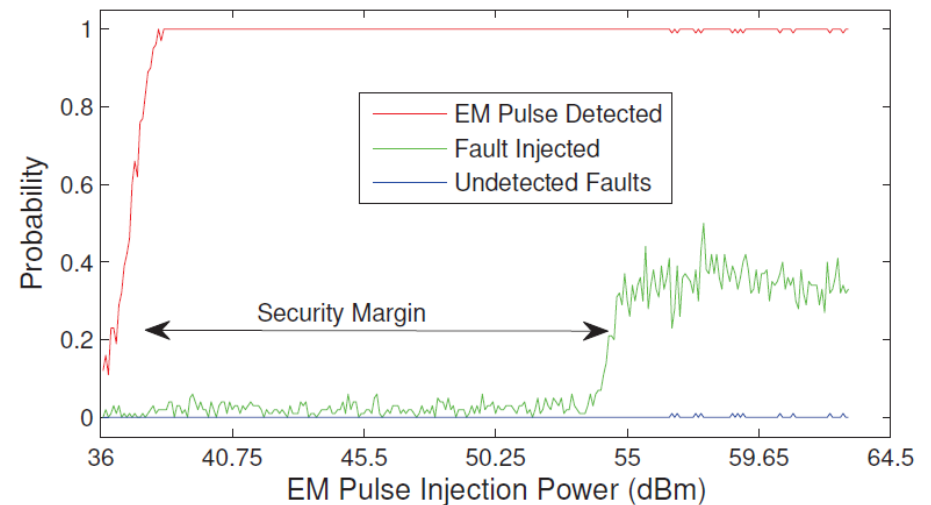
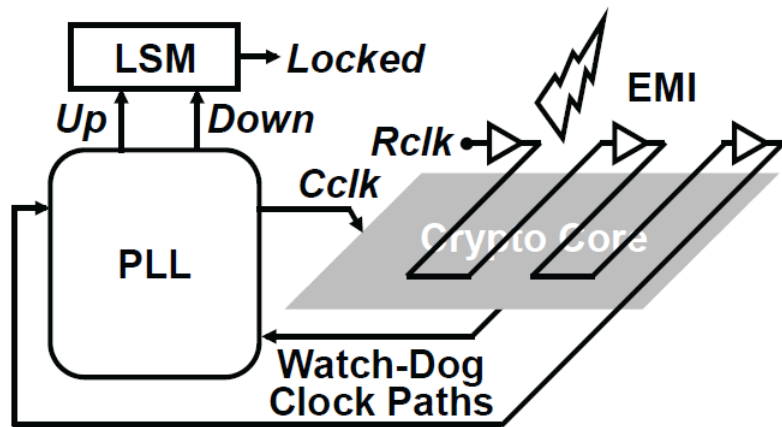
- **In-situ – 100% detection rate**
- Local Environment – false-positive/false-negative
- Global Environment – false-positive/false-negative



Toshinori Sato, Yuji Kunitake:
A Simple Flip-Flop Circuit for Typical-Case Designs for DFM.
ISQED 2007: 539-544

EMFI Sensor

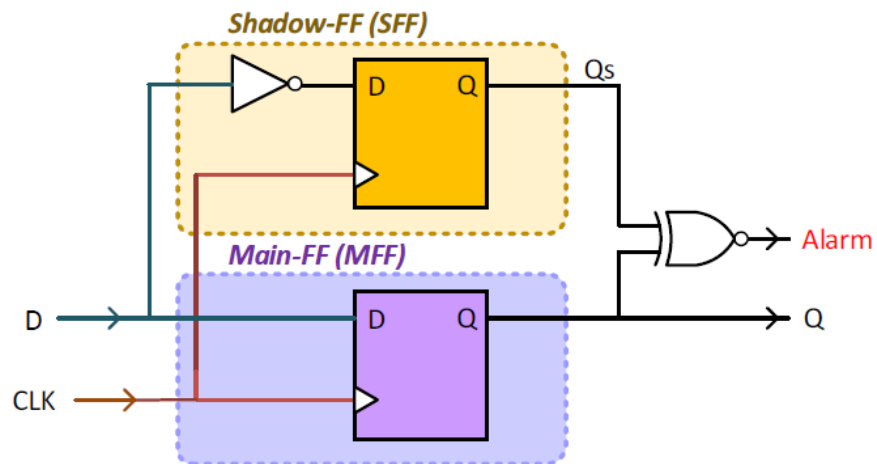
- Local, based on PLL lock-detection



Noriyuki Miura, Zakaria Najm, Wei He, Shivam Bhasin, Xuan Thuy Ngo, Makoto Nagata, Jean-Luc Danger: PLL to the rescue: a novel EM fault countermeasure. DAC 2016: 90:1-90:6

EMFI Sensor

- In-situ, based on Redundant State

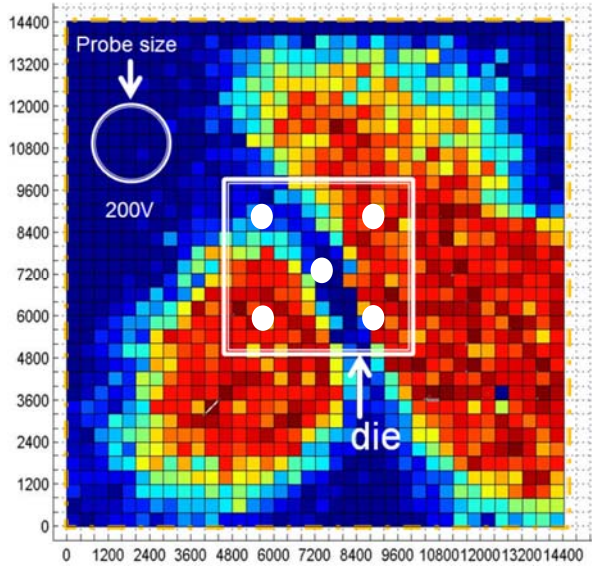
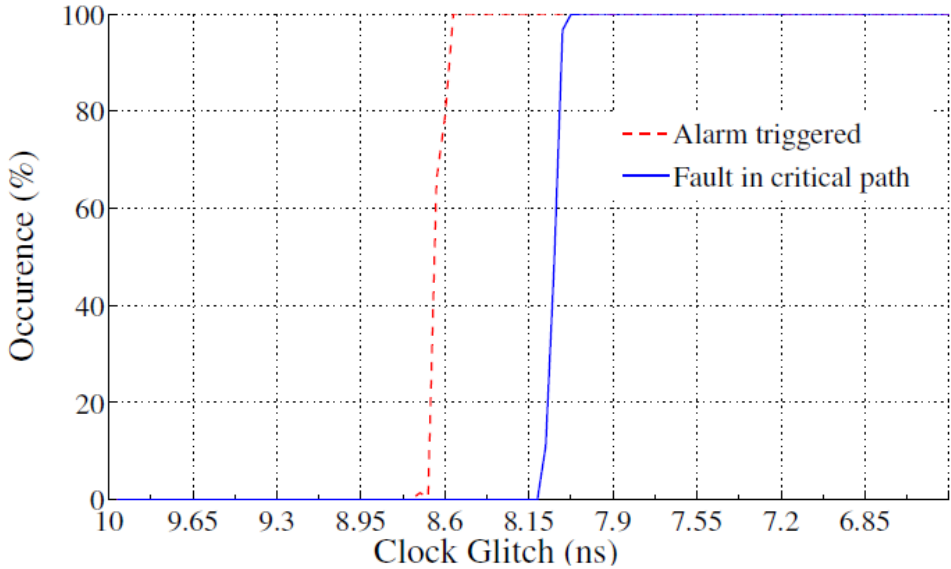
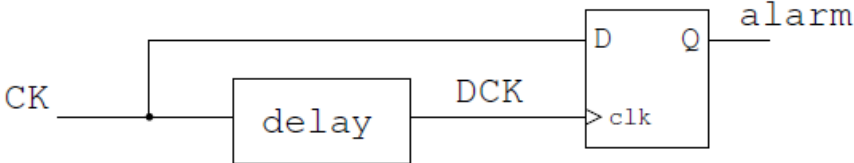


- In-situ, 100% detection rate

C. Deshpande, B. Yuce, P. Schaumont and L. Nazhandali, "Employing Dual-complementary Flip-Flops to Detect EMFI Attacks," 2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Beijing, CN, October 2017.

EMFI Sensor

- Local, based on Delay

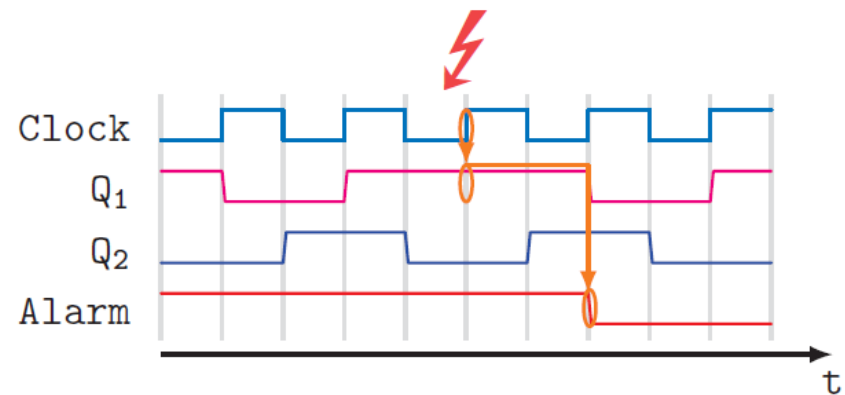
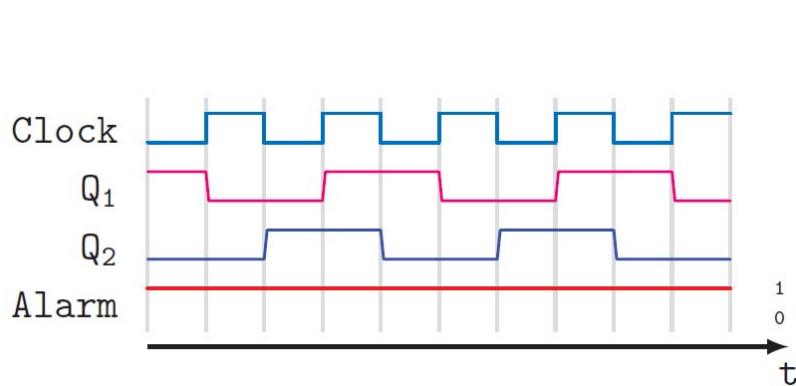
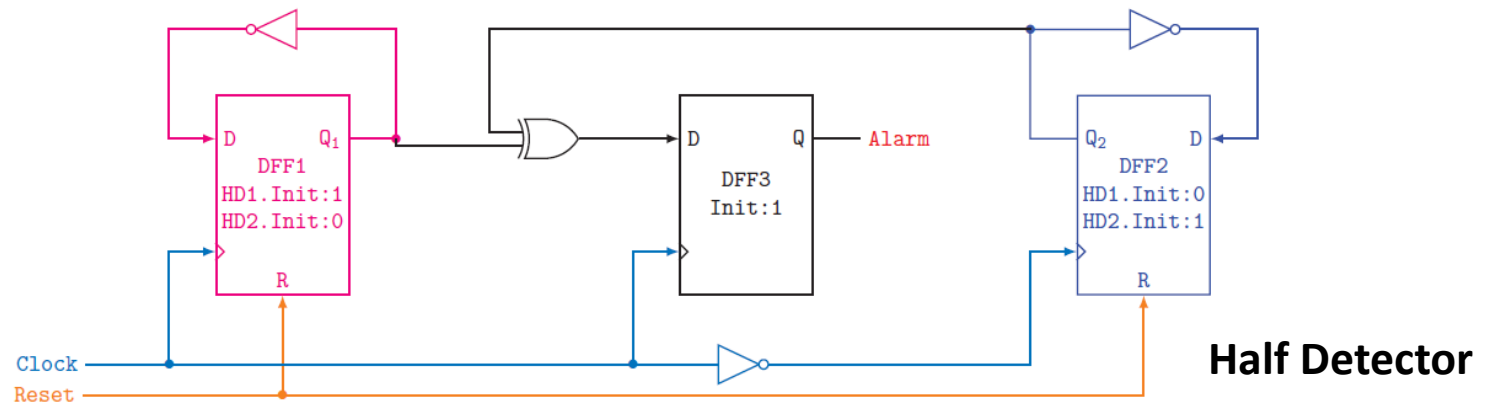


Xilinx Spartan Cartography, 5 sensors

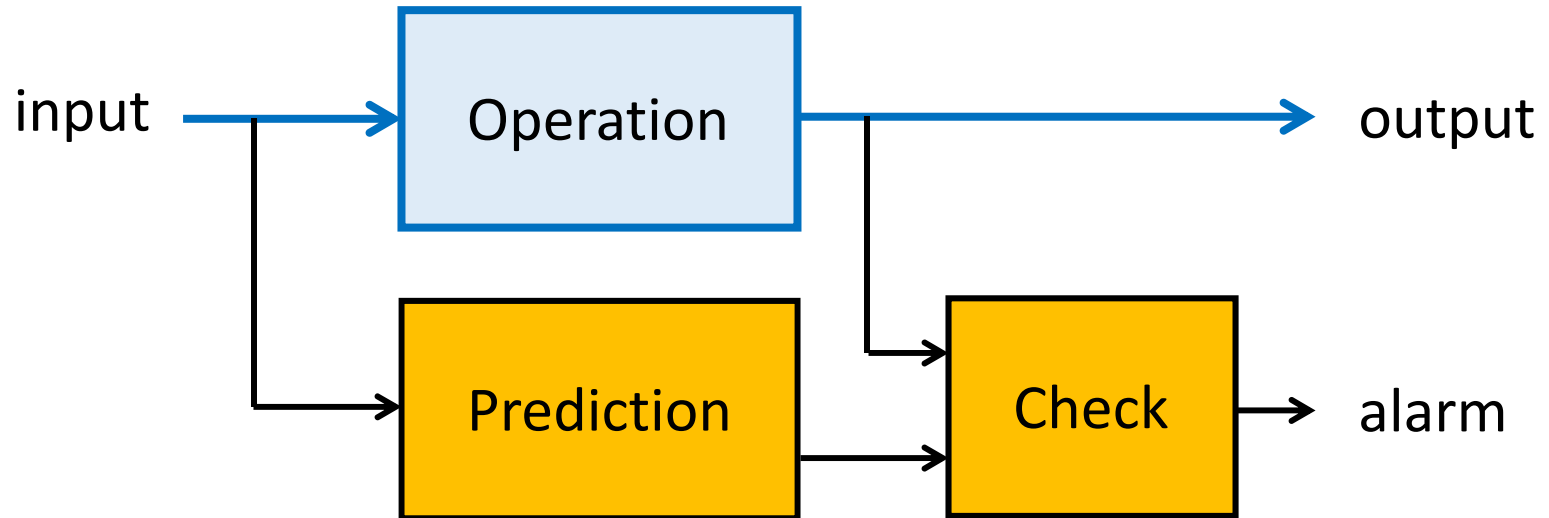
Loïc Zussa, Amine Dehbaoui, Karim Tobich, Jean-Max Dutertre, Philippe Maurine, Ludovic Guillaume-Sage, Jessy Clédière, Assia Tria: Efficiency of a glitch detector against electromagnetic fault injection. DATE 2014: 1-6

EMFI Sensor

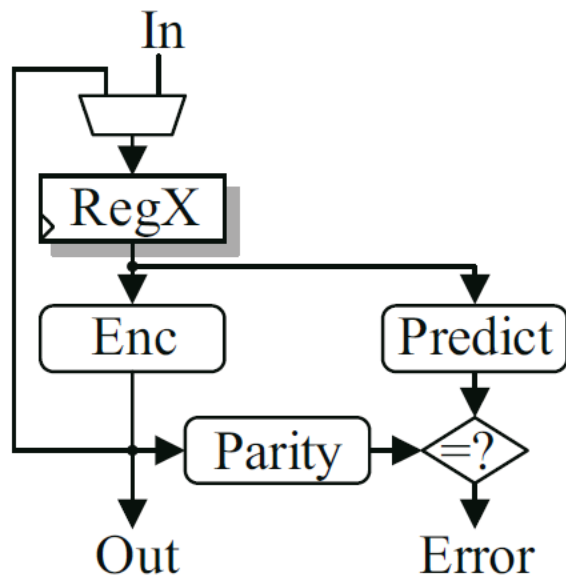
- Local, based on Complementary Clock Signals



Concurrent Error Detection



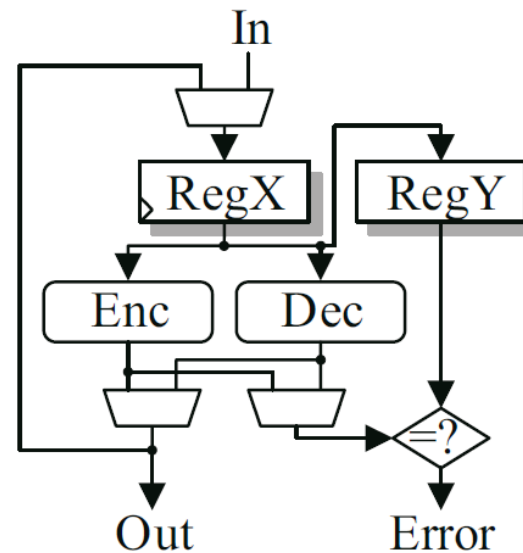
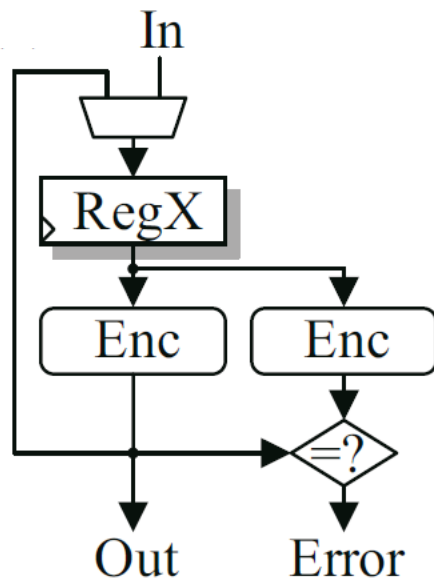
CED on Encryption



- Information redundancy
- Predict smaller than Encoder, therefore error coverage is imperfect
- Linear Parity Prediction is easy: $\text{parity}(a \text{ xor } p) = \text{parity}(a) \text{ xor } \text{parity}(p)$
- Non-linear Prediction harder: $\text{parity}(\text{sbox}(a))$

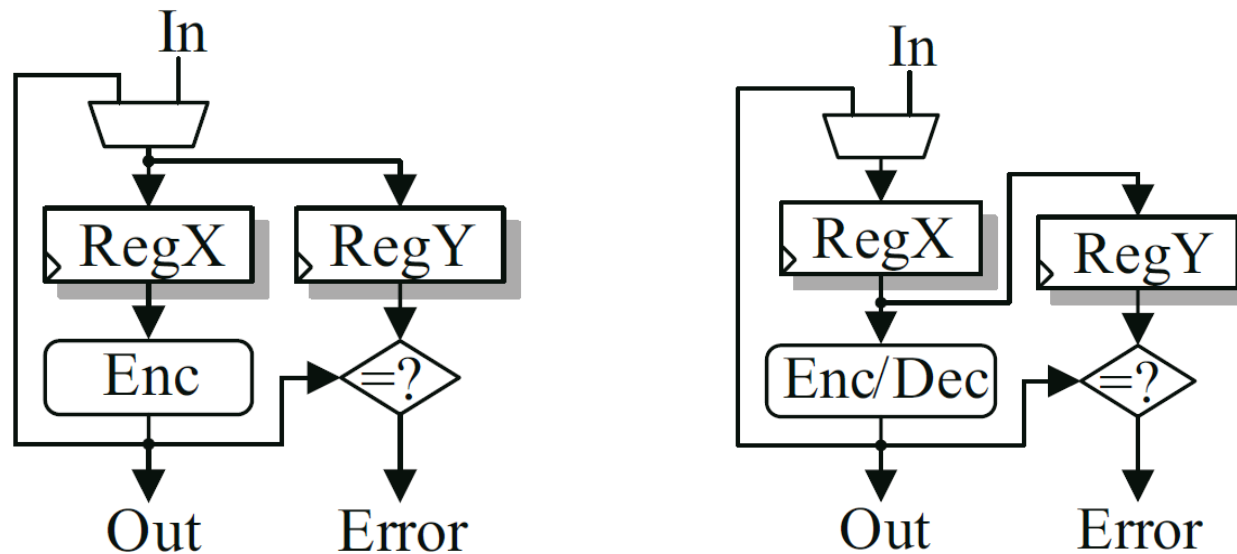
CED on Encryption

- Spatial redundancy
- Full coverage, but expensive in area



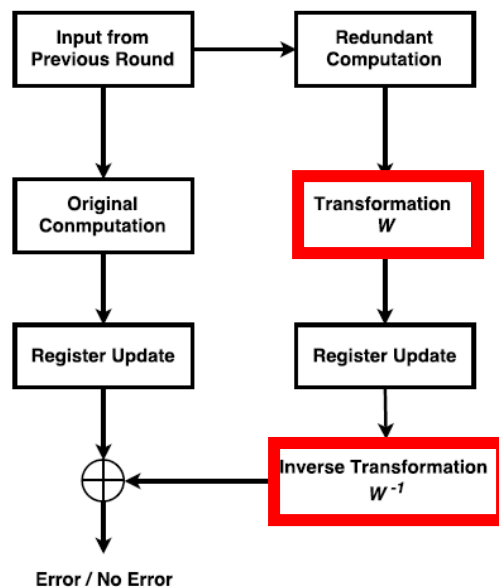
CED on Encryption

- Time redundancy
- Full coverage, but expensive in performance



CED Risks

- Time/Spatial redundancy susceptible to redundant fault injection
- Fault Collision probability in redundant copies increase for small, biased faults (1-bit, 2-bit, ..)
- Addressed by *Fault space transformation*



S. Patranabis et al: A Generic Approach to Counter Differential Fault Analysis and Differential Fault Intensity Analysis on AES-Like Block Ciphers. IEEE Trans. Information Forensics and Security 12(5): 1092-1102 (2017)

Algorithm-specific Countermeasures

- Elliptic Curve Cryptosystems

$$E: y^2 + a_1.x.y + a_3.y = x^3 + a_2.x^2 + a_4.x + a_6$$

$$P: (x, y)$$

$$Q = k.P$$

- Point validity check
- Curve integrity check
- Coherence check (eg. Montgomery Powering Ladder)

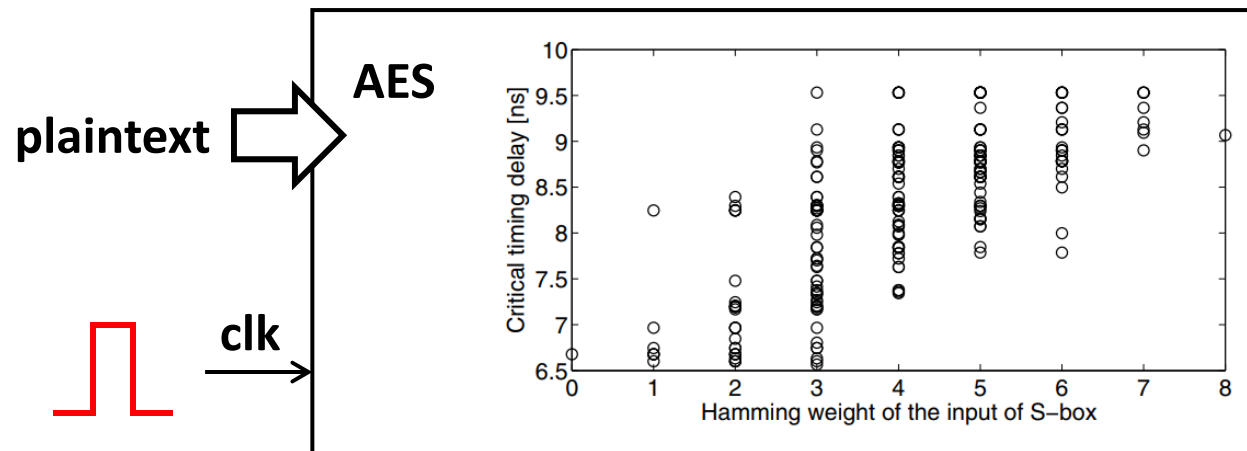
Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, Ingrid Verbauwhede:
State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and
Countermeasures. HOST 2010: 76-87

Outline

1. Taxonomy of Countermeasures
2. Fault Prevention
3. Fault Detection
4. Fault Response

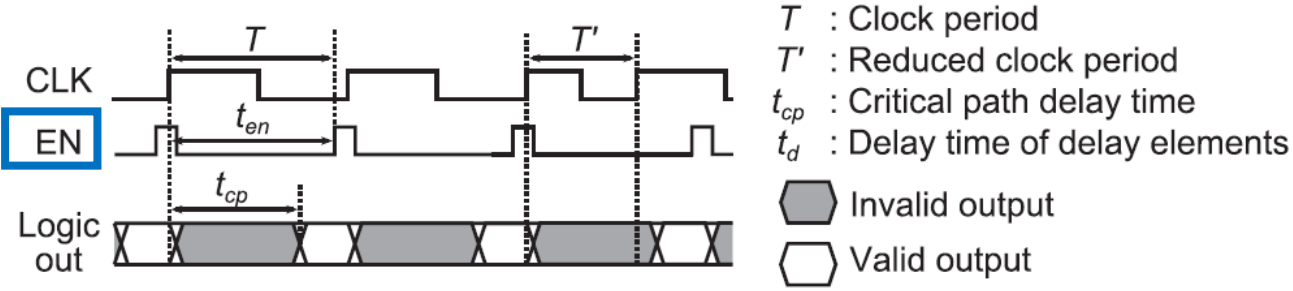
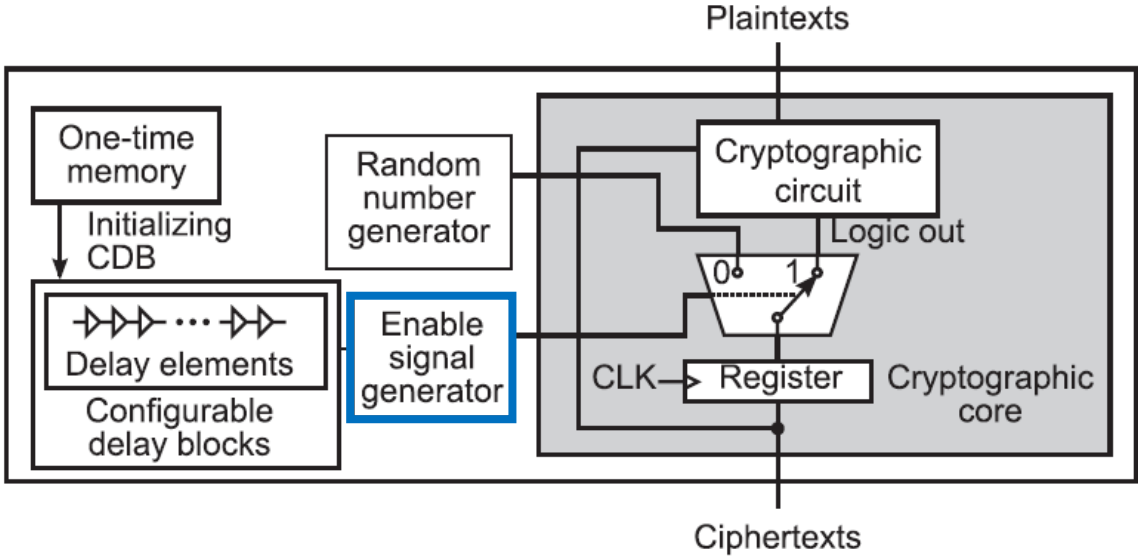
Fault Response

- Often ignored, but a crucial aspect of countermeasure design
- The response *itself* may be exploited in an attack
 - Eg. In FSA, fault response leads to a hypothesis test



Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, Kazuo Ohta:
Fault Sensitivity Analysis. CHES 2010: 320-334

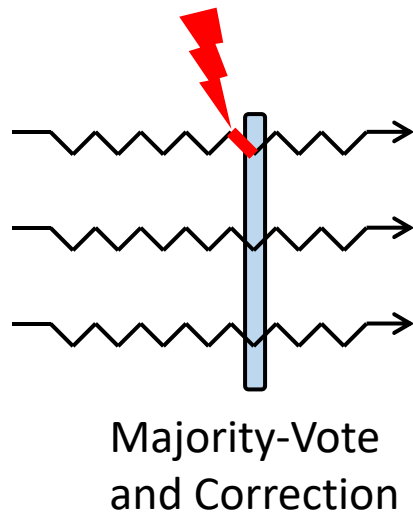
Fault Sensitivity Countermeasure



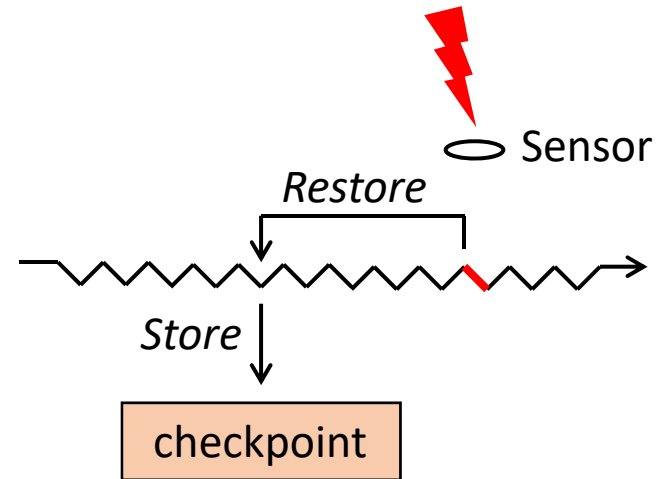
Sho Endo, Yang Li, Naofumi Homma, Kazuo Sakiyama, Kazuo Ohta, Daisuke Fujimoto, Makoto Nagata, Toshihiro Katashita, Jean-Luc Danger, Takafumi Aoki: A Silicon-Level Countermeasure Against Fault Sensitivity Analysis and Its Evaluation. IEEE Trans. VLSI Syst. 23(8): 1429-1438 (2015)

Fault Response: Redundancy vs Checkpoint-Restore

Concurrent Error Detection



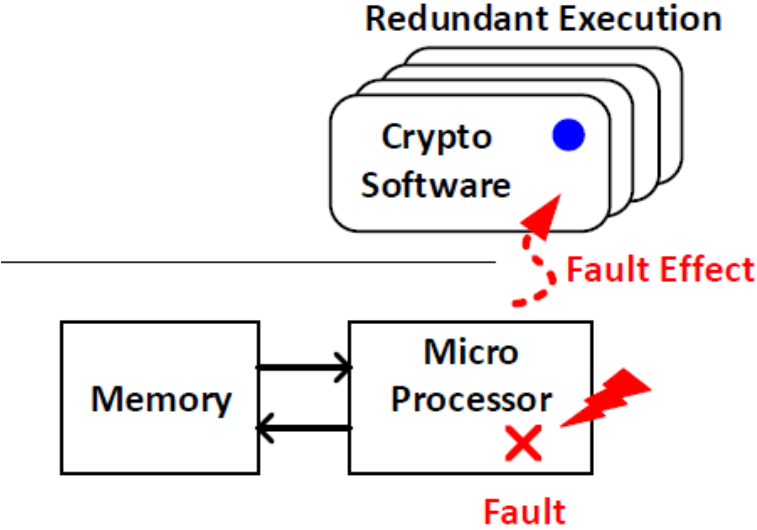
Checkpoint-Restore



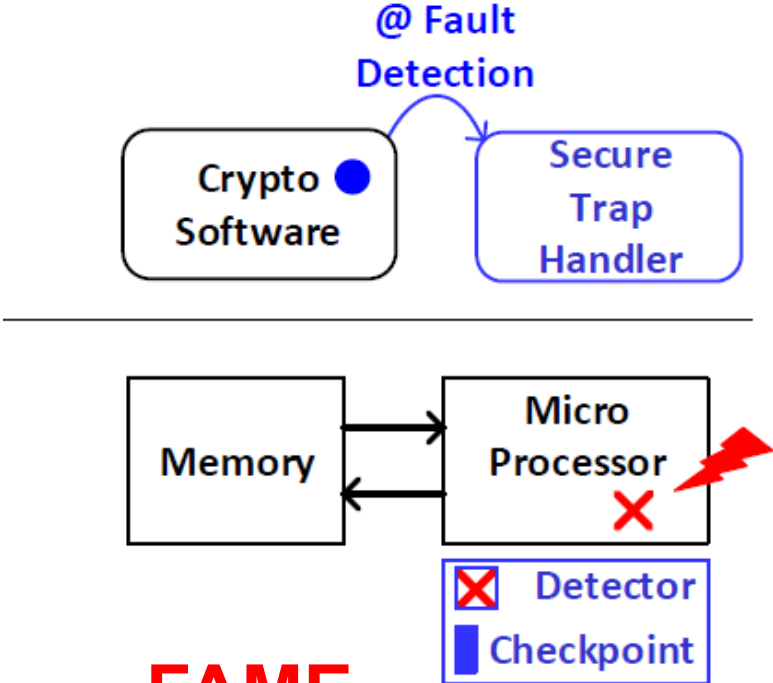
Redundancy vs Checkpoint-Restore

Example: Protecting Embedded Software against Fault Attacks

Concurrent Error Detection

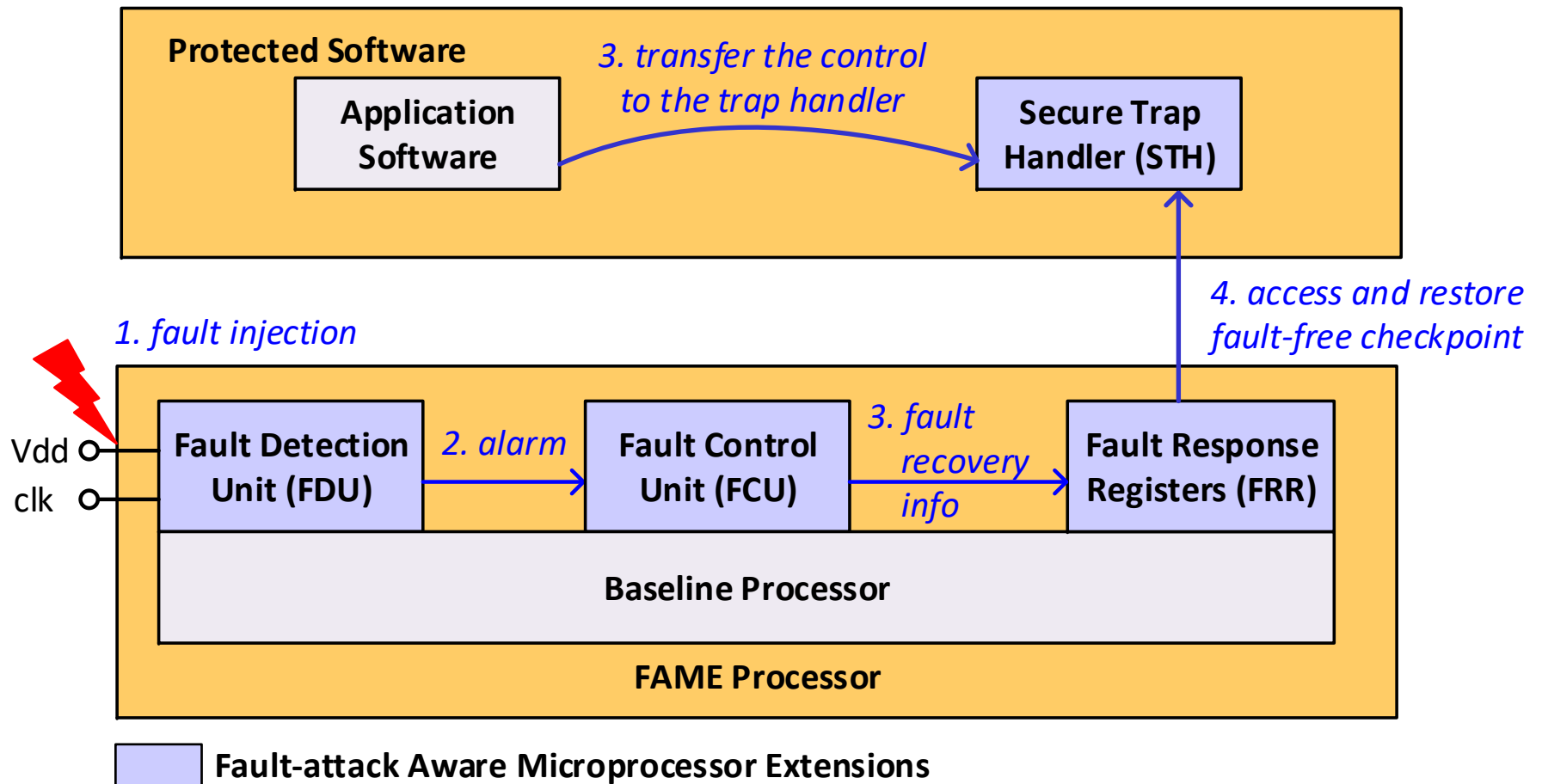


Checkpoint-Restore



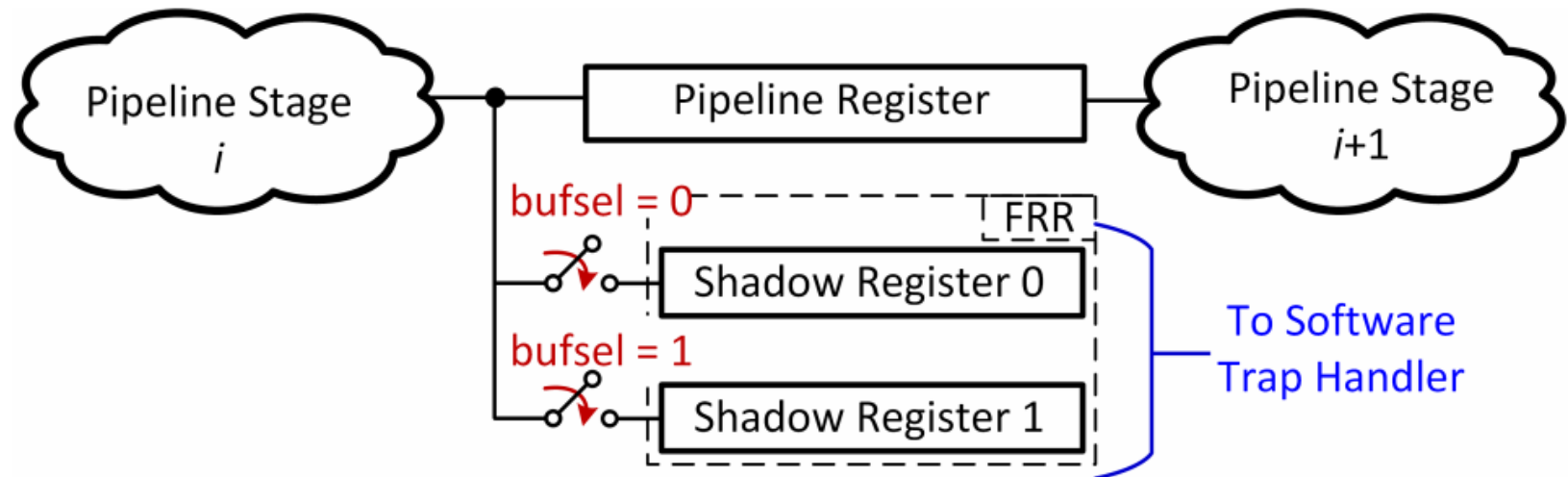
FAME
Fault-attack Aware
Microprocessor Extension

FAME Operation [HASP 16]

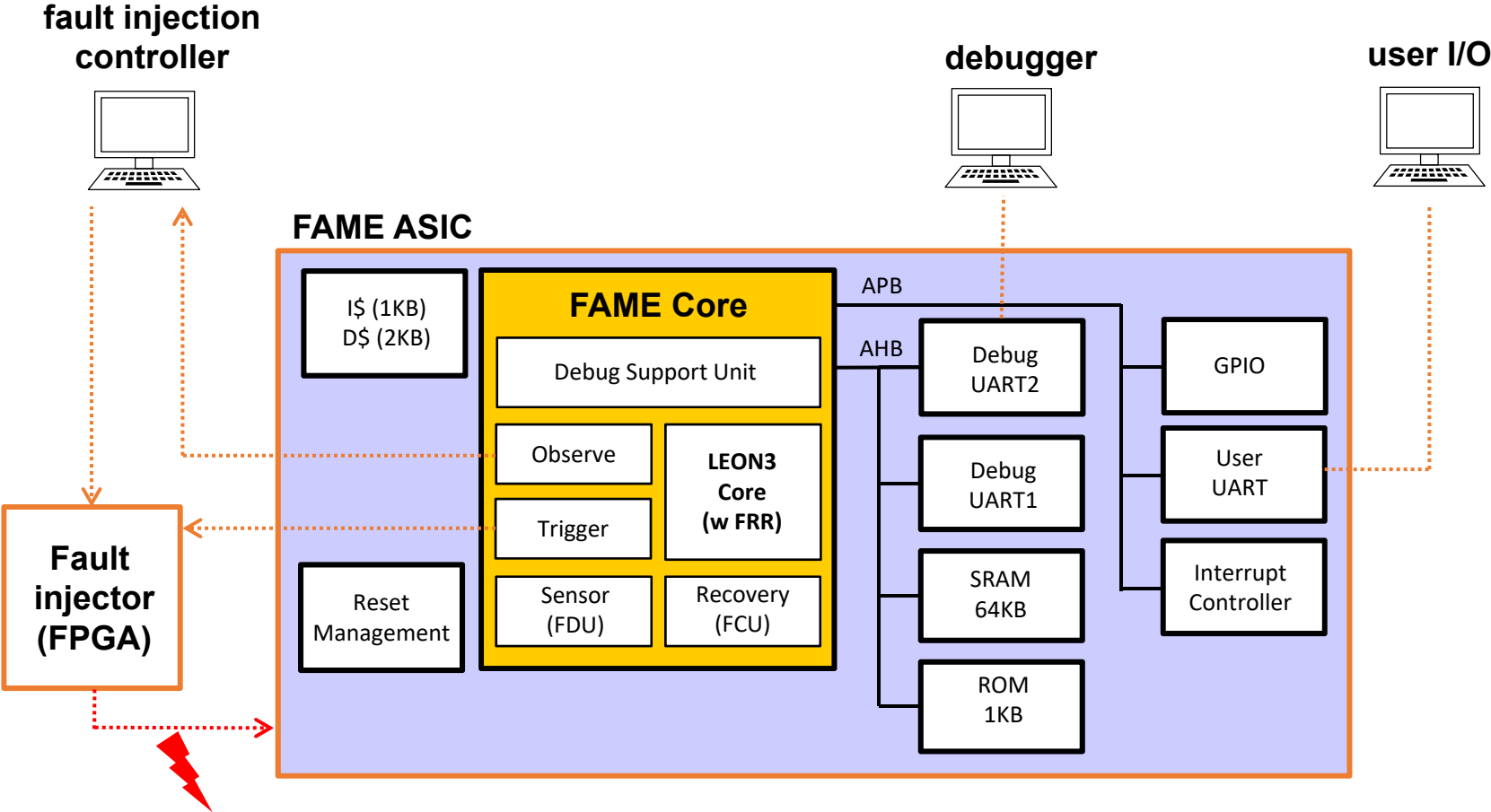


Single-cycle Checkpointing Hardware

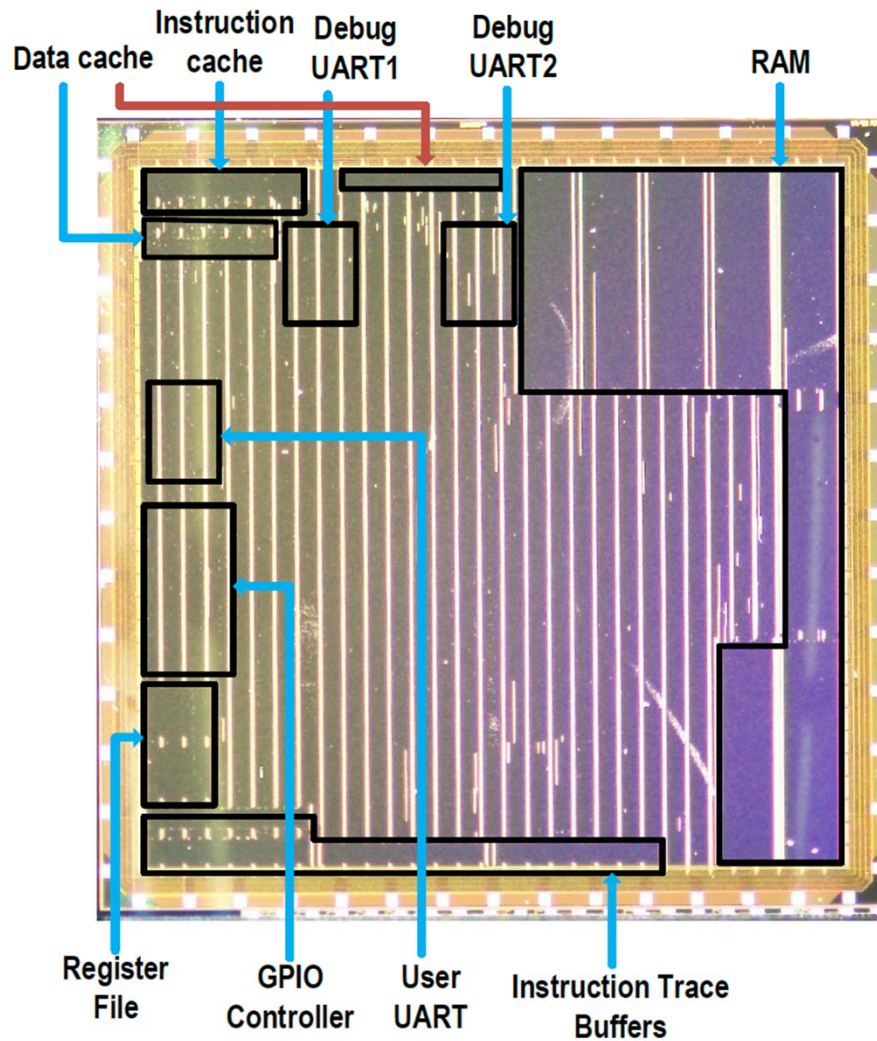
- Fault Response Registers (FRR) for critical processor state, including PC, PSR and last two pipeline stages



FAME Chip 1 Block Diagram

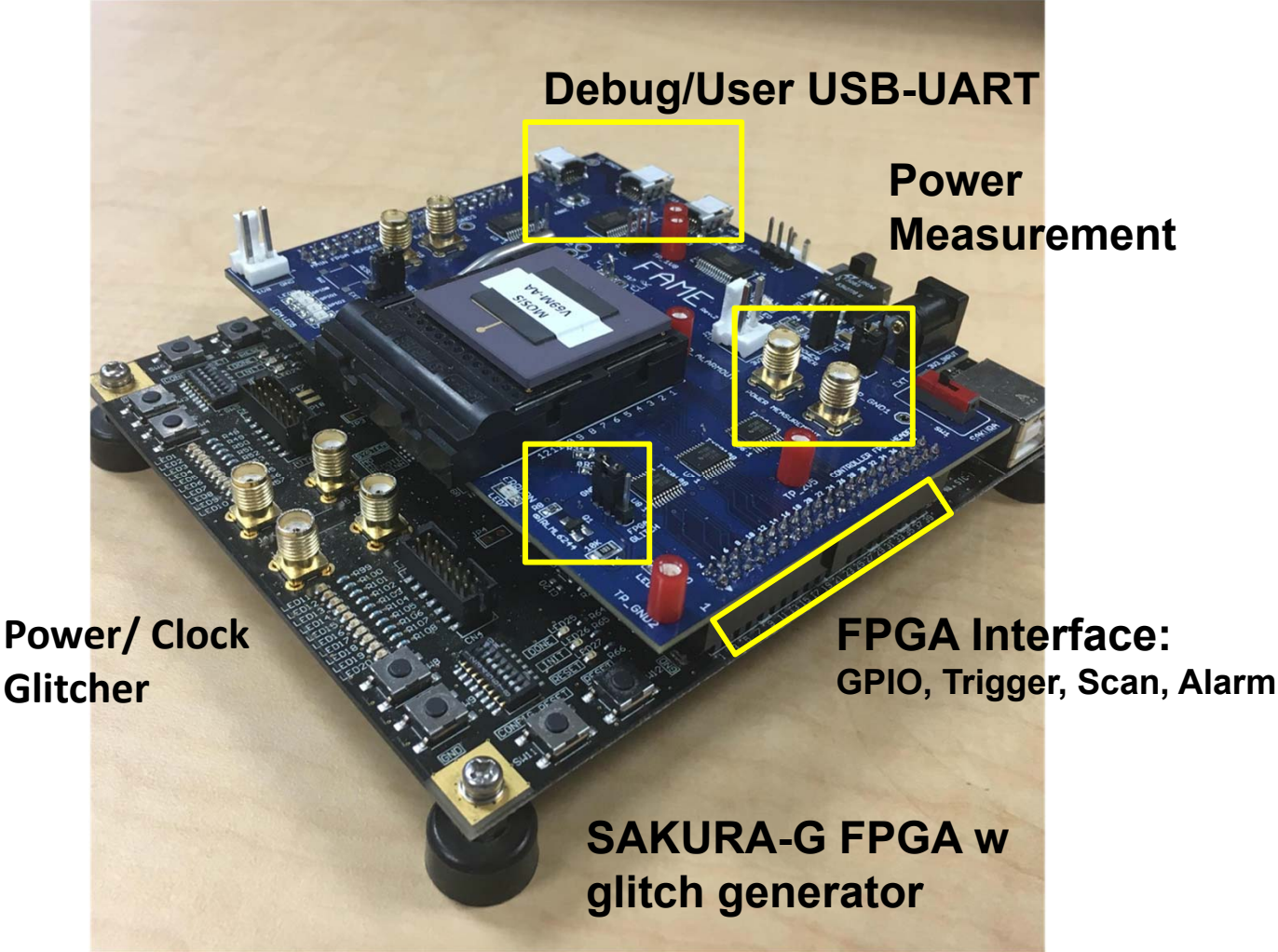


FAME Chip 1 Micrograph

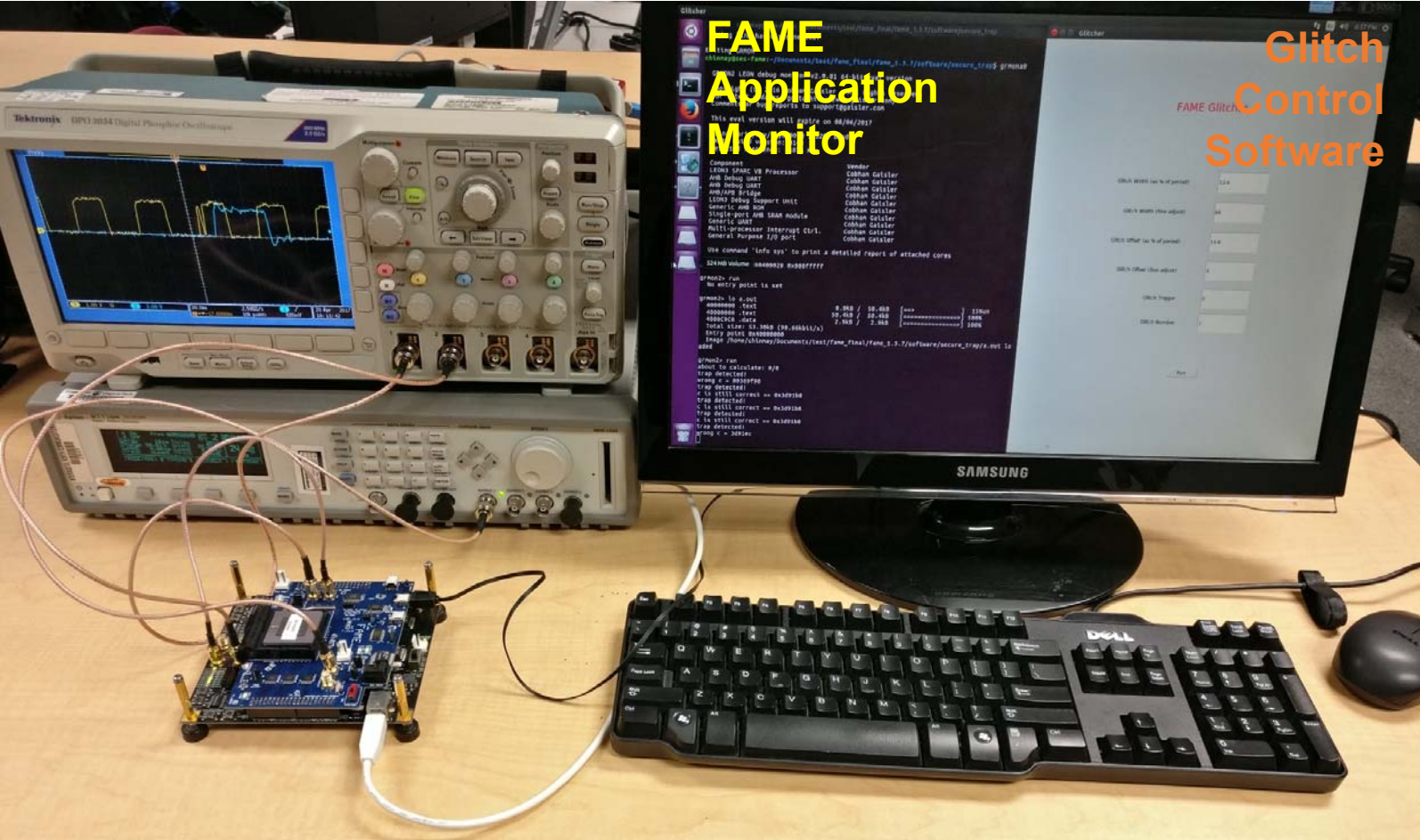


- 180nm 6LM TSMC
- 25 mm² die area
- Active area
 - LEON3: 6.217mm²
 - w FAME: 6.301 mm²
 - w FAME+Diag: 6.364 mm²
- FAME extensions overhead 1.35% (of active area)
- 80 MHz clock
- 54 I/O
 - Clock, reset
 - 8 I/O, 16 Core Power
 - 3x UART
 - 4 GPIO
 - 4 Trigger
 - Sensor alarm monitor
 - Scan and test pins
- 108-pin PGA package

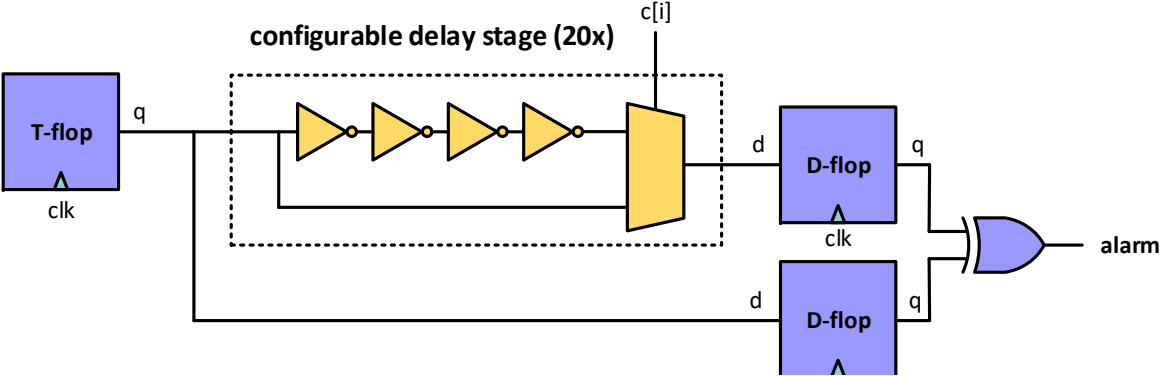
FAME Chip 1 Test PCB



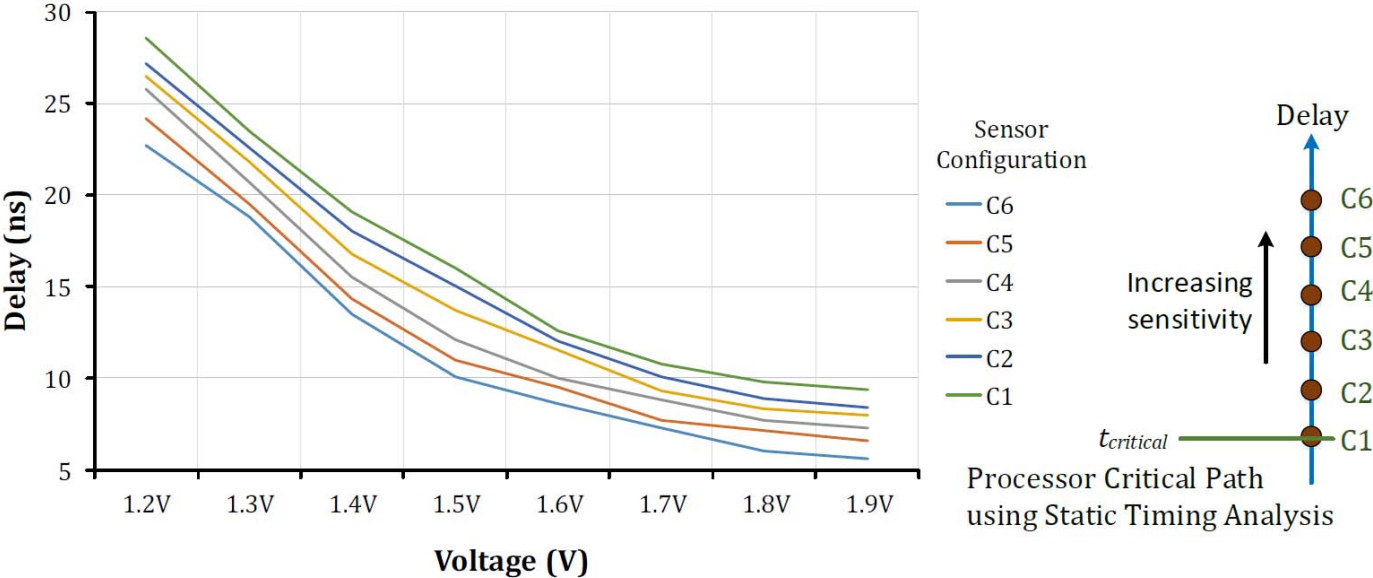
FAME Chip 1 Test Setup



FAME Chip 1 Fault Sensor



Fault Detection Sensitivity



Secure Trap Handler Development

```
int ptc = 3; //Pin Try Counter
char devicePIN[5] = "12824";

int VerifyPin(userPIN) {
    ptc--;
    if (ptc > 0)
        if (Cmp(userPIN, devicePIN))
            result = 1;
        else
            result = 0;
        ptc--;
    else
        result = 0;
    return result;
}
```

```
call 40001f5c <Cmp()>
nop
mov %o0, %g1
cmp %g1, 0
be <else branch>
```

```
<if_branch>:
mov 1, %g1
stb %g1, [ %fp + -2 ]
b <end_of_Cmp()>
```

```
<else_branch>:
clrb [ %fp + -2 ]
ldub [ %fp + -1 ], %g1
add %g1, -1, %g1
stb %g1, [ %fp + -1 ]
b <end_of_Cmp()>
```

falls through

FAME Based Design

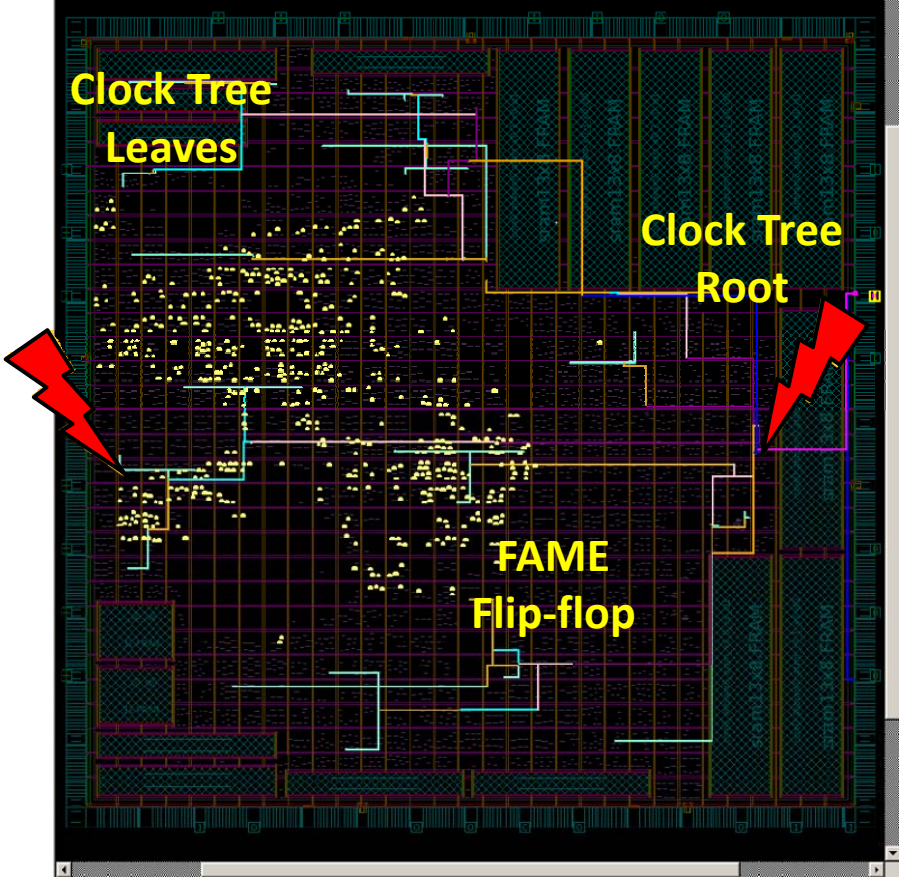
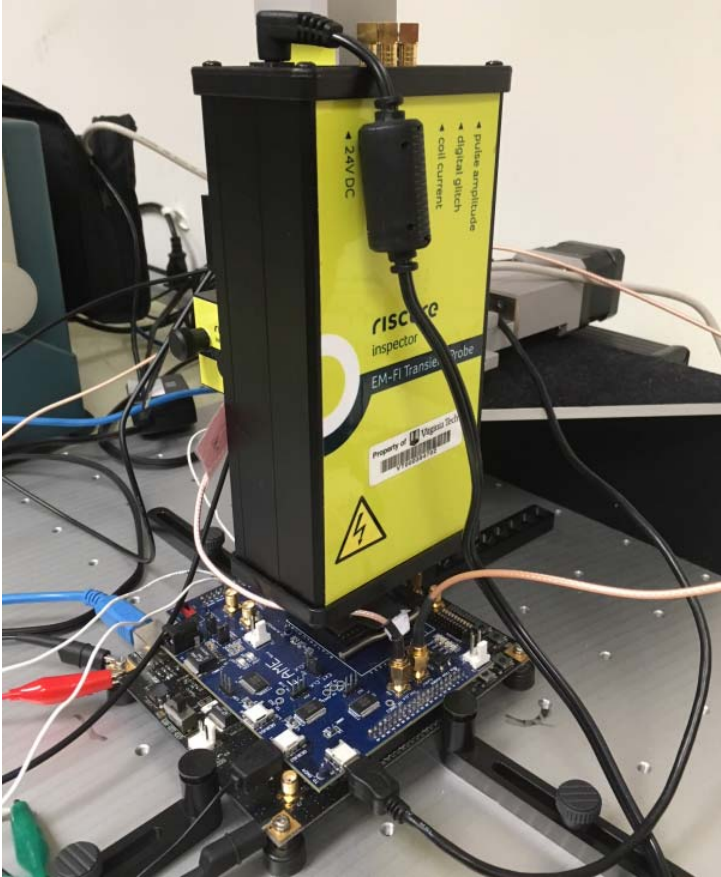
```
int ptc = 3; //Pin Try Counter
char devicePIN[5] = "12824";
int noFault = 1;
int VerifyPin(userPIN) {
    if (ptc > 0)
        if (Cmp(userPIN,devicePIN))
            result = noFault;
        else
            result = 0;
            ptc--;
    else result = 0;
    return result;
}
```

```
SecureTrapHandler() {
    if (ptc > 0)
        ptc--;
        noFault = 0;
}
```

Outline

1. Taxonomy of Countermeasures
2. Fault Prevention
3. Fault Detection
4. Fault Response

EMFI on FAME



[DAC2018]

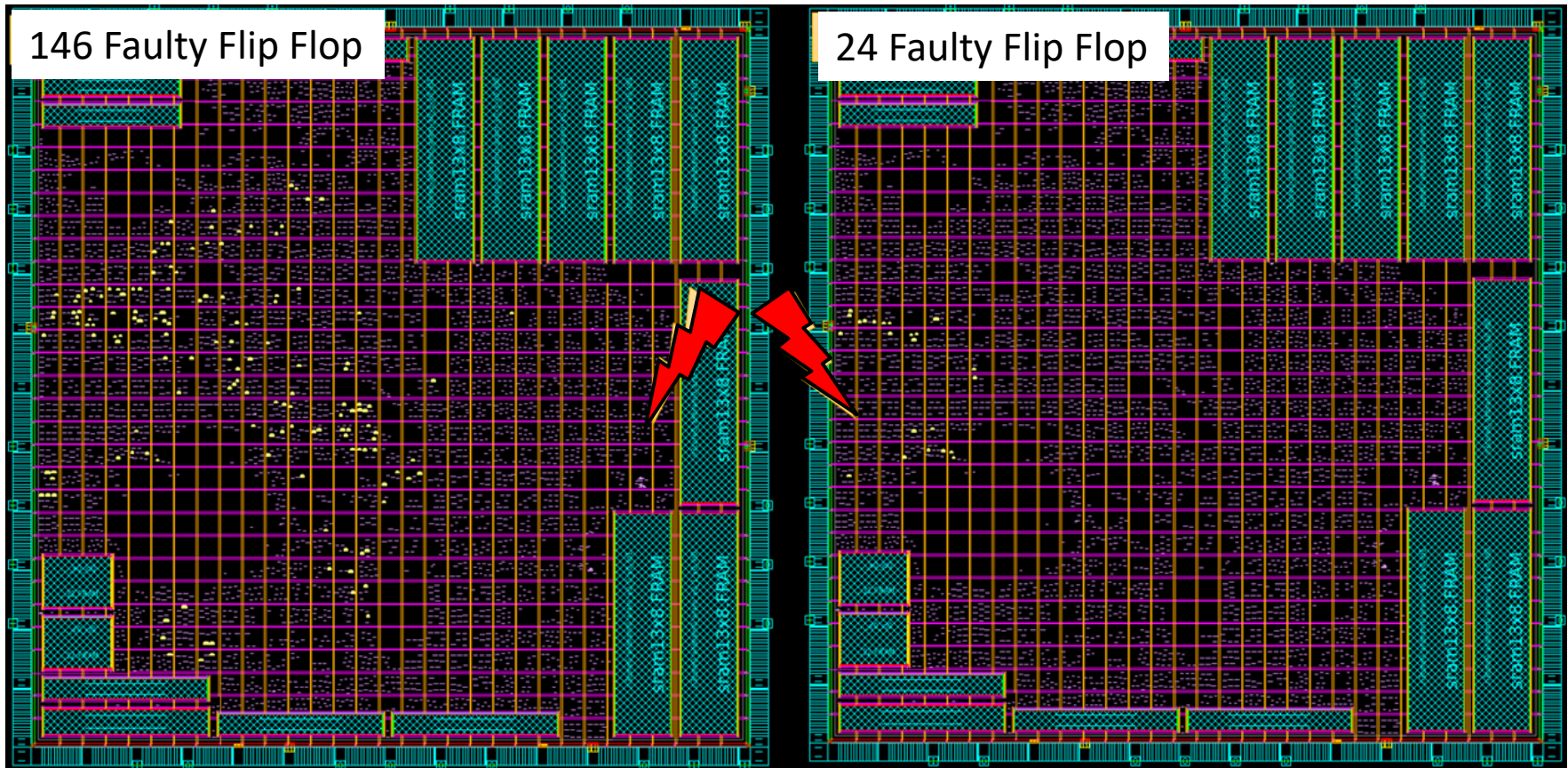
EMFI on FAME

Global Effect of EMFI

Injection at clock tree root

Local Effect of EMFI

Injection at clock tree leaves

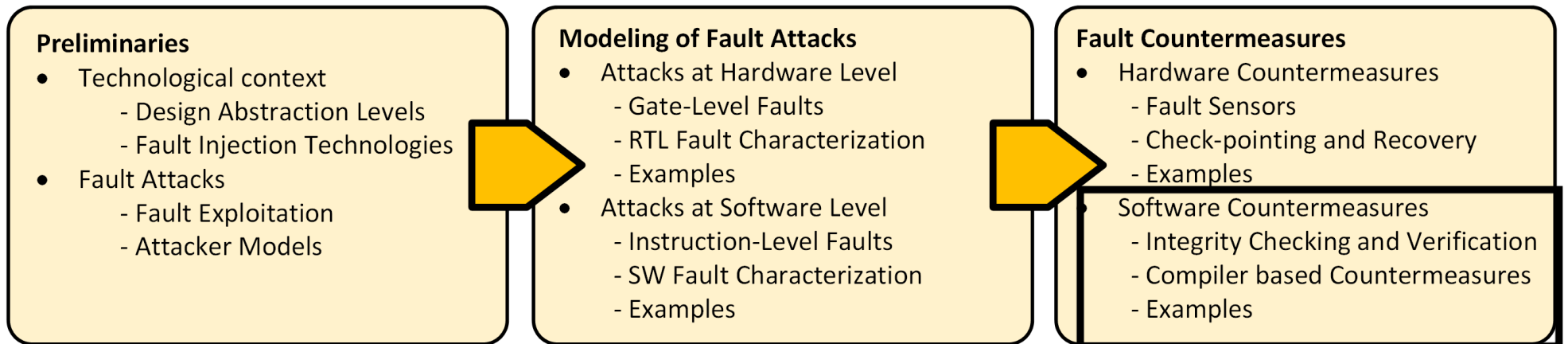


[DAC2018]

Software Countermeasures against Fault Attacks

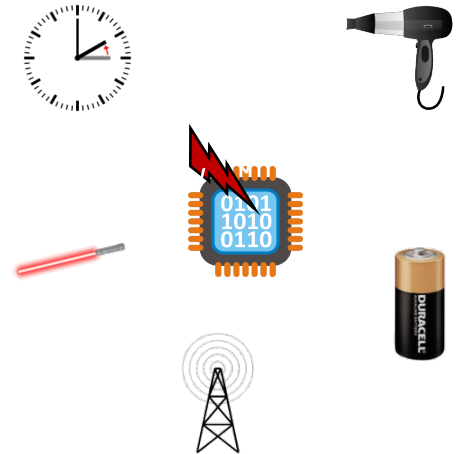
Karine Heydemann





Protections against fault injection attacks

- **Hardware-based countermeasures** [El Bar et al., 2006]
 - Light sensor, glitch detectors [Zussa et al., 2014]
 - Redundancy [Karaklajic et al, 2013]
 - Error correcting codes (registers, memory)
- **Too expensive for small devices and no full guaranty**
- **Software-based countermeasures** [Verbauhede, 2011] [Rauzy et al., 2015]
 - Redundancy at function level
 - Algorithm-specific protection (e.g. RSA)
 - Ad-hoc protections designed by expert engineers
- In practice combination of both in secure elements



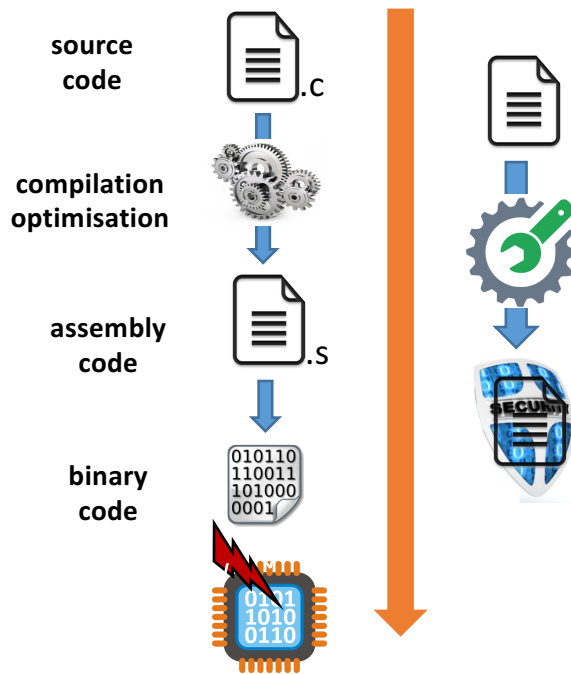
SW protections against fault injection attacks

- **Manually added**
 - Tedious, error-prone
 - Highly expensive
 - Expertise needed
- **Need for automation and capitalization**
 - Cost reduction, availability for non-experts
 - Adaptable to a specific product
 - Trade-off between security and performance
- **Need for generic protections**
 - Not dedicated to a class of algorithms (crypto)
 - Against different fault models / attacker expertise



Software protection against fault attacks

Code hardening



▪ At which code level?

- **Source**
 - Code review, portability, independent from tools
- **Compilation**
 - Adaptability and/or control over code optimization
- **Assembly**
 - Final “attacked code”, low level information available
- **Binary**
 - Global view, availability of library codes

→ Multiple needs



Outline

- **Principle of software countermeasures**
 - Data integrity
 - Code integrity
 - Control-flow integrity
- **Compiler-assisted code hardening**
 - Protection against instruction skip
 - Loop hardening

Countermeasures for data integrity

Fault model

- Data corruption

Redundancy-based protections

- Duplication of instructions involved in the computation
- Comparison of results of both computations
- Detection of
 - Register corruption (r1 or r2)
 - Load corruption
- Need for available registers

```
add r1, r0, #1
```

duplicate
and
compare

```
add r1, r0, #1
add r2, r0, #1
cmp r2, r1
bne fault_detection
```

```
ldr r1, [r0]
```

duplicate
and
compare

```
ldr r1, [r0]
ldr r2, [r0]
cmp r2, r1
b.ne fault_detection
```



A. Barenghi et al. *Countermeasures against fault attacks on software implemented AES*.
5th Workshop on Embedded Systems Security (WESS'10)

Countermeasures for data integrity

Fault model

- Data corruption

Redundancy-based protections

- Data duplication in addition to instruction duplication
- Detection of
 - Memory corruption
 - Load corruption
 - Register corruption
- High overhead: performance and memory footprint

```
ldr r1, [r0]
```

Duplicate data,
instruction,
and compare

```
ldr r1, [r0]
ldr r2, [r0+offset]
cmp r2, r1
b.ne fault_detection
```



Reis et al. *SWIFT: Software Implemented Fault Tolerance*.
International Symposium on Code Generation and Optimization. 2005

Countermeasures for code integrity

Fault model

- Instruction corruption

Redundancy-based protections

- Instruction duplication with detection
- Detection of
 - One instruction skip
 - Some instruction replacements

```
ldr r1, [r0]
```

duplicate
and
compare

```
ldr r1, [r0]
ldr r2, [r0]
cmp r2, r1
b.ne fault_detection
```



A. Barenghi et al. *Countermeasures against fault attacks on software implemented AES*.
5th Workshop on Embedded Systems Security (WESS'10)

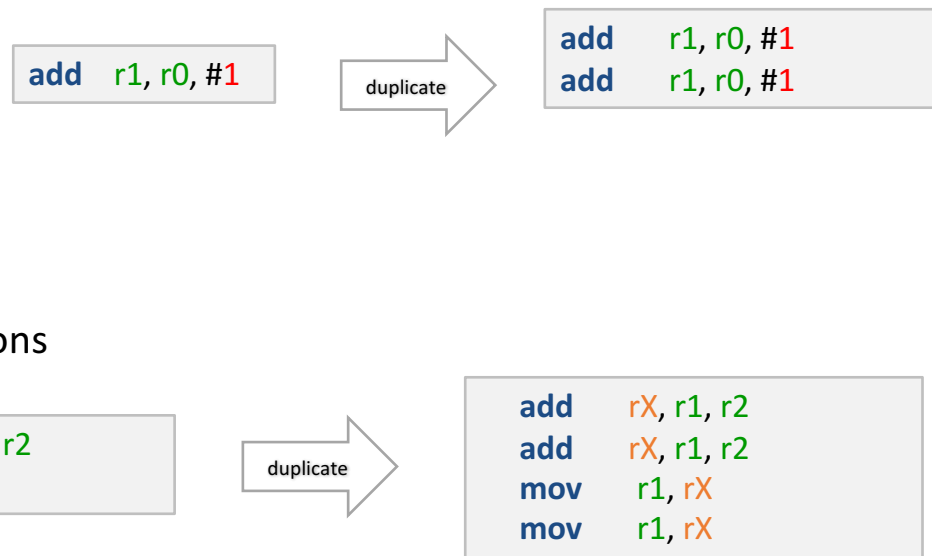
Countermeasures for code integrity


Fault model

- Instruction skip

Redundancy-based protections

- Instruction duplication without detection
 - Tolerance to one instruction skip
 - Only for idempotent instructions
 - Transformation of non-idempotent instructions



 Moro et al. *Formal verification of a software countermeasure against instruction skip attacks.* Journal of Cryptographic Engineering 2014.

Countermeasures for code integrity

Fault model

- Instruction skip

Redundancy

- Instruction
- Tol
- On
- Tra

```
add r1, r0, #1
```

No software protection for full code integrity
(i.e. against all kinds of instruction replacement or disruption)

```
add r1,
```

instructions

```
mov r1, rX
```



Moro et al. *Formal verification of a software countermeasure against instruction skip attacks*.
Journal of Cryptographic Engineering 2014.

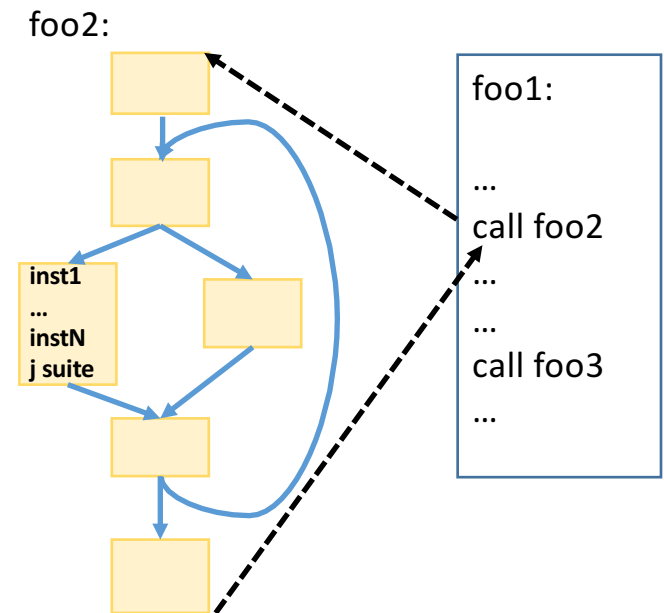
Control flow integrity

Fault model

- Jump insertion

Different levels of control-flow integrity

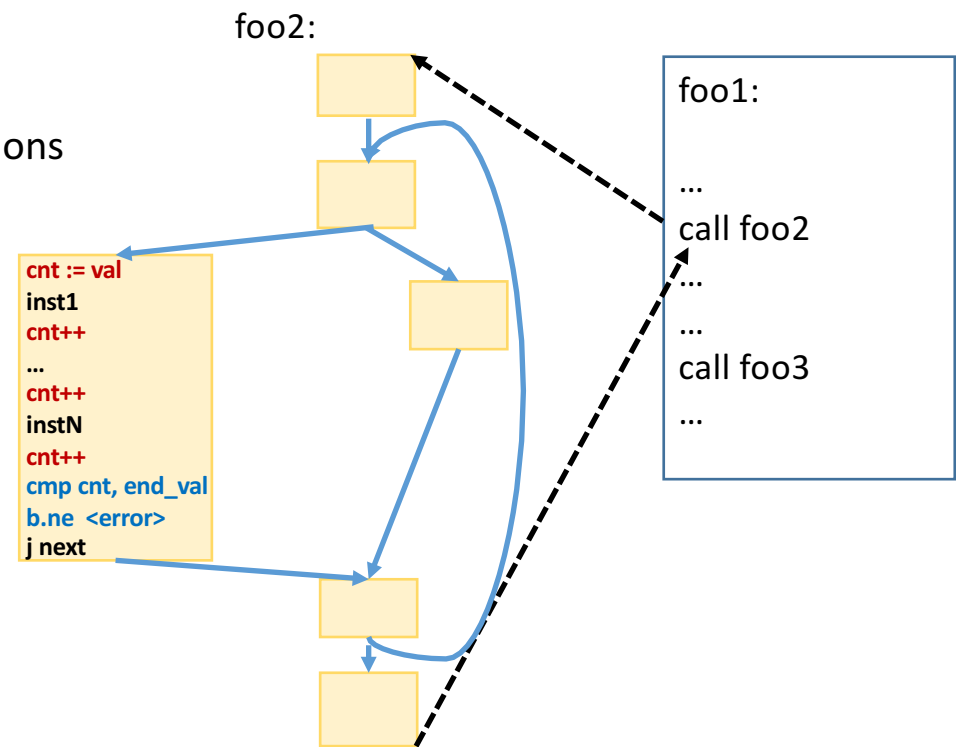
- Intra basic block
integrity of straight-line code
- Intra procedural
integrity of control flow transfers inside a function
(control flow graph)
- Inter procedural
integrity of function calls and returns



Countermeasures for control flow integrity

Counter-based protections [Akkar et al., 2003]

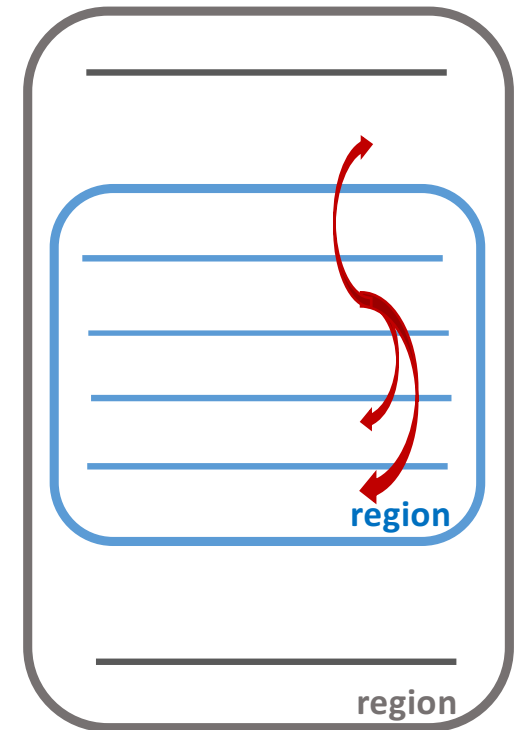
- Dedicated counters incremented between instructions
- Check of their values at some specific points
- Intra basic block scheme
 - Detection of intra basic block jumps



Countermeasures for control flow integrity

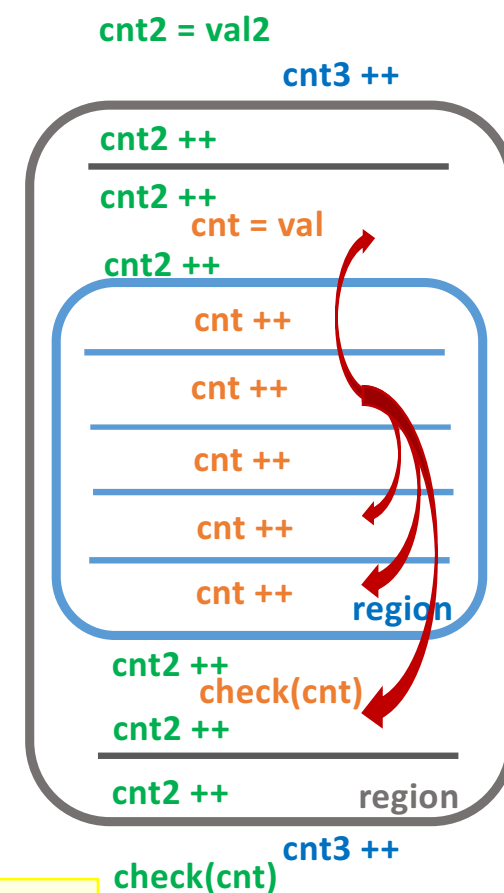
Counter-based protections

- Protection scheme for C control-flow constructs
- **Fault models**
 - Jump insertion
 - Test inversion
- **Objective**
 - All statement must be executed
 - in the right order
 - as expected according to the execution context
 - Or an attack must be detected



Countermeasures for control flow integrity

- **Region-based protection scheme:** straight-line regions, if-then-else constructs, switch constructs, loops with or without early exit or continue, function calls
- Each region has its own **protection counters** and may use **extra variables** to hold the condition values that influence the control flow (e.g. loop exit condition)
- **Nesting and overlapping** of the protection of regions in order to guarantee that an attack will eventually be detected



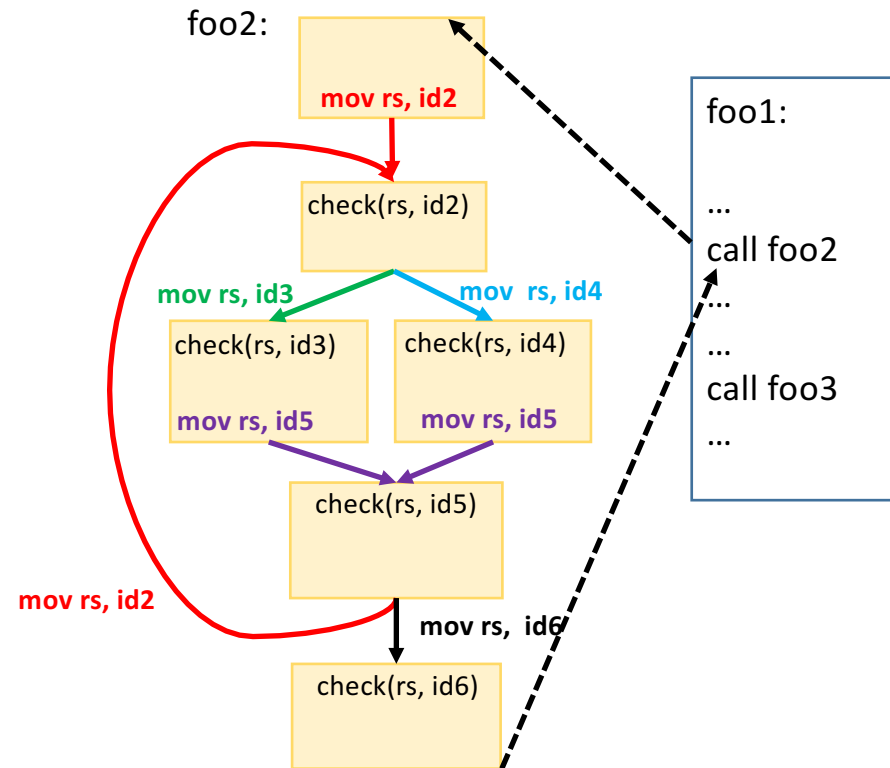
Countermeasures for control flow integrity

Signature-based protections [Oh et al. 2002] [Goloubeva et al., 2005]

- Identifiers are assigned to basic blocks (and functions)
- Use to check every single control flow transfer
- Global signature computation enables to limit the number of checks
- Only protect control flow transfers

Combination [SIED, 2003]

- Step counters inside basic blocks and signature for control flow transfers

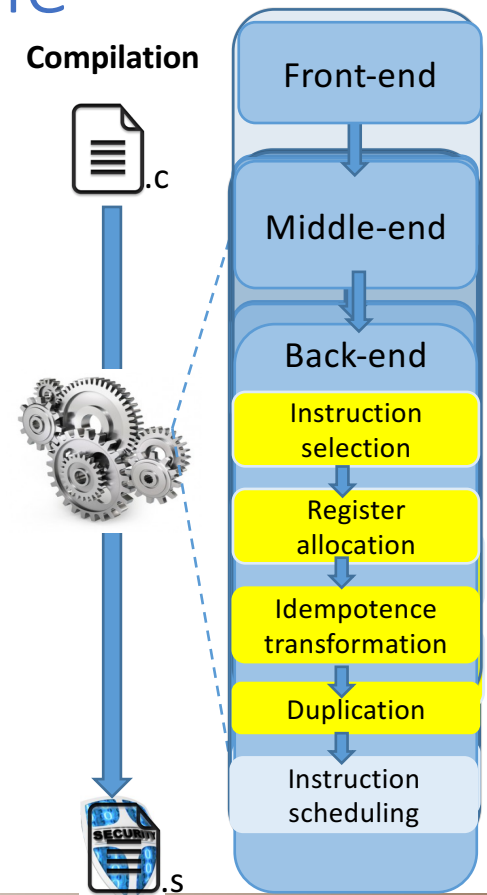


Outline

- **Principle of software countermeasures**
 - Data integrity
 - Code integrity
 - Control-flow integrity
- **Compiler-assisted code hardening**
 - Protection against instruction skip
 - Loop hardening

Protection at compilation-time

- **Protection scheme against instruction skip** [Moro et al. 2014]
- Main principle: **duplication of idempotent instructions**
- **Take advantage of compilation flow to**
 - Force the generation of idempotent instructions
 - Modification of the instruction selection
 - Modification of the register allocation
 - Additional transformation for remaining non-idempotent instructions (e.g. push and pop instruction that use and modify the stack pointer)
 - Add an instruction duplication pass
 - Let the scheduler optimize the duplicated code
- Automatically protected code with better code size and performance



T. Barry et al. *Compilation of a Countermeasure Against Instruction-Skip Fault Attacks*. CS2 2016.

Compile-time loop hardening

Motivation

- Several attacks exploit a **corruption of loop iteration count** (early or deferred exit)
 - Buffer overflows [Nashimoto et al. 2017]
 - Cryptanalysis by round reduction [Dehbaoui et al. 2013, Espitau et al. 2016]
 - Authentication process [Dureuil et al., FISSC, 2016]

Considered fault model

- One instruction skip
- Or one general register corruption
- During loop execution

```
void aes_addRoundKey_cpy(  
    uint8_t *buf, uint8_t *  
    key, uint8_t *cpk)  
{  
    register uint8_t i = 16 ;  
  
    while (i--)  
    {  
        buf[i] ^= key[i] ;  
        cpk[i] = key[i] ;  
        cpk[16+i] = key[16+i] ;  
    }  
}
```



Loop hardening scheme

Goal

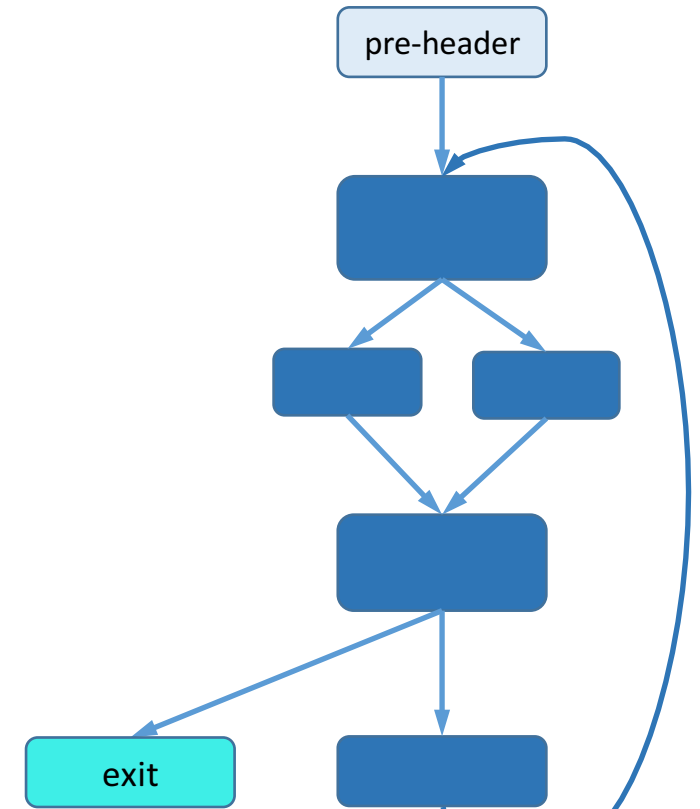
- The loop performs the right iteration count and exit from the right exit, or an attack is detected

Protection principle

- For each loop exit, check its outcome

Realisation

- **Duplication** of all the instructions involved in the computation of an exit condition
- Addition of **verification basic blocks** on all the paths following from an exiting block
- Protection of the **internal control flow** that may impact an exit condition



Loop hardening scheme

Goal

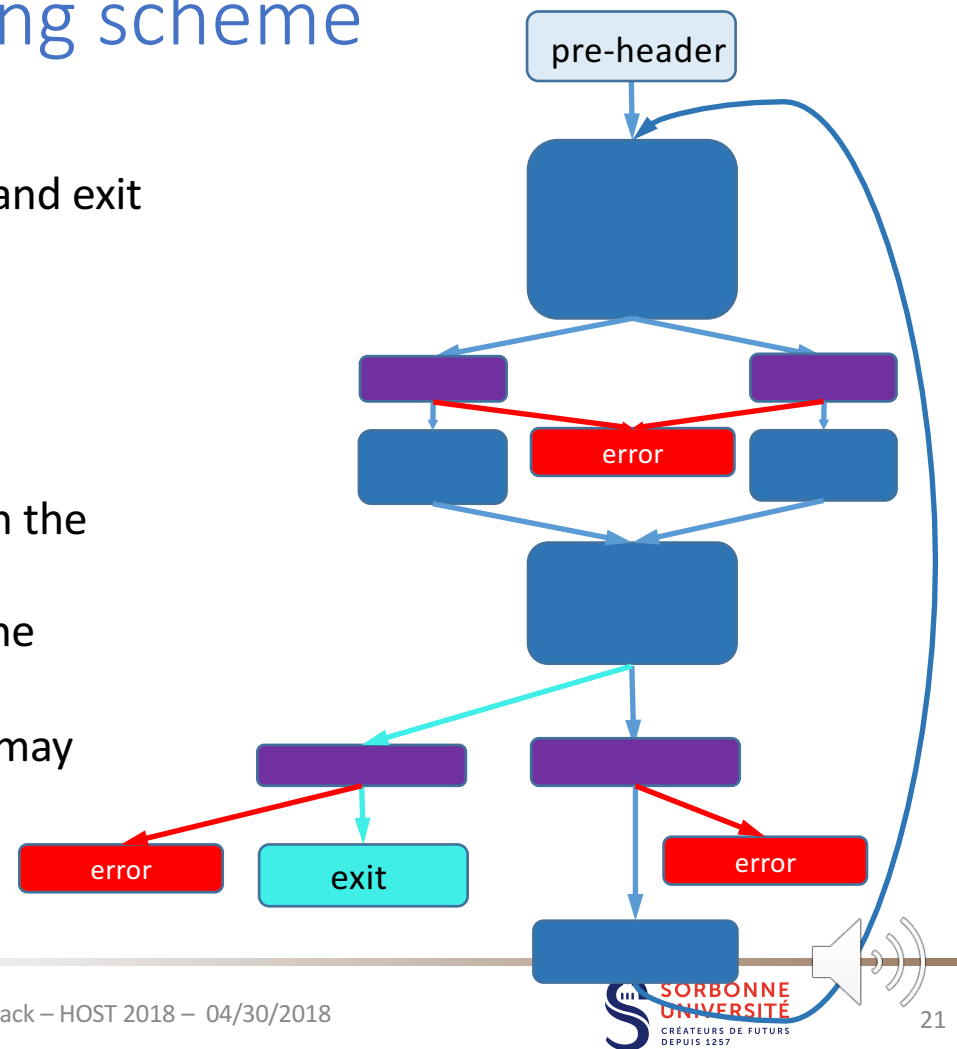
- The loop performs the right iteration count and exit from the right exit, or an attack is detected

Protection principle

- For each loop exit, check its outcome

Realisation

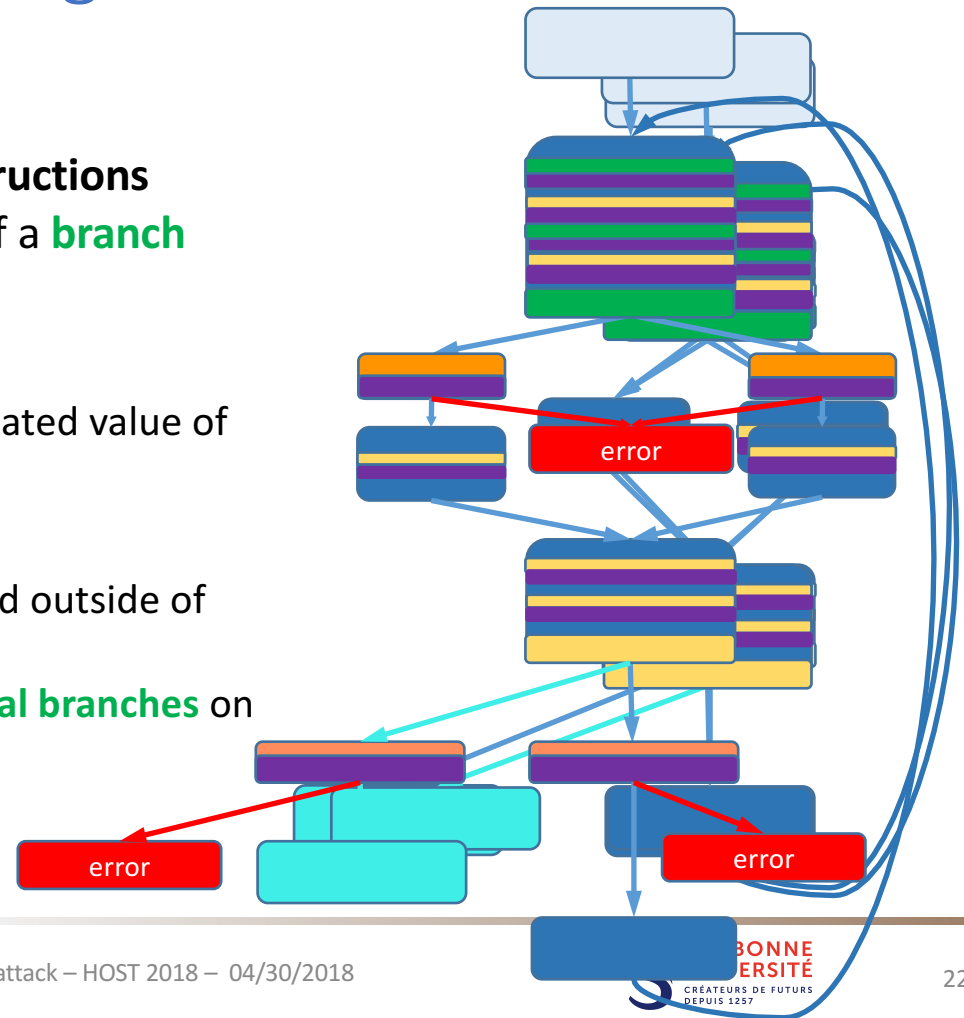
- **Duplication** of all the instructions involved in the computation of an exit condition
- Addition of **verification basic blocks** on all the paths following from an exiting block
- Protection of the **internal control flow** that may impact an exit condition



Loop hardening scheme

For each exit of a loop

- Determination by a backward analysis of the **instructions involved** in an **exit condition** or in an condition of a **branch that may influence an exit condition**
- **Instruction duplication**
 - Creation of a **second data flow** leading to a duplicated value of the condition, independant from the original one
- **Addition of verification blocks**
 - Checks of **the duplicated exit condition** inside and outside of the loop to verify the exiting branch
 - Checks of **the duplicated conditions of the internal branches** on all possible following paths
 - **Call to a fault detection handler**



Loop hardening pass and a compilation flow

Automaton and insertion in a compilation flow

- Implemented in a compiler (LLVM 3.9+) at the intermediate level
 - independence from the target architecture
- Insertion after optimisation passes that may alter the protection

Experimental results

- 99% of harmful simulated fault are detected

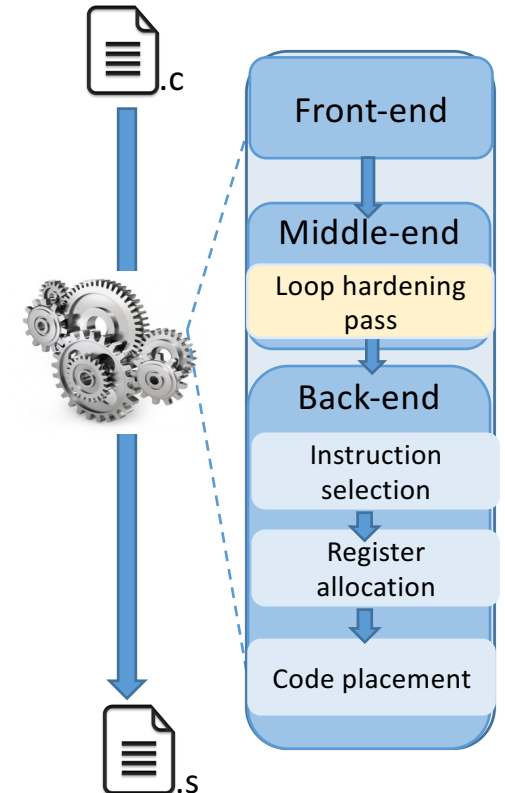
Harmful post-securing transformations and optimisations

- All kind of redundancy elimination
- Instruction selection, register allocation, code placement optimisation

→ Compiler is not compliant with protection / security properties

→ **Need to analyse the generated code**

→ **Need to deactivate, adapt, or add some passes to ensure the security property**



Summary and conclusion

- **Various types of protection**
 - Large set of fault models / attacker capabilities
- **Need of automatic code hardening** and against a large set of (faults) attacks
 - Compiler-assisted code hardening
 - Framework enabling the analysis and the **preservation of security properties**
 - In the compilation flow
 - For a post-compilation robustness analysis
- **Combination of protections**
 - Interaction between protections? Stacking or smarter combination?



References

- [Dehbaoui et al. 2013] Amine Dehbaoui, A., Mirbaha, A-P., Moro, N., Dutertre, J-M., Tria, A.: Electromagnetic Glitch on the AES Round Counter. COSADE_2013: 17-31
- [Dureuil et al, FISSC, 2016] Dureuil, L., Petiot, G., Potet, M.-L., Crohen, A., and Choudens, P. D. (2016). FISSC: a Fault Injection and Simulation Secure Collection. In Proceedings of International Conference on Computer Safety, Reliability and Security, volume 9922 of (SAFECOMP), pages 3–11, Trondheim, Norway. Springer Berlin / Heidelberg.
- [El Bar et al., 2006] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. (2006). The sorcerer’s apprentice guide to fault attacks. Proceedings of the IEEE, 94(2):370–382.
- [Espitau et al. 2016] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, Mehdi Tibouchi: Loop-Abort Faults on Lattice-Based Fiat-Shamir and Hash-and-Sign Signatures. SAC 2016: 140-158
- [Karaklajic et al, 2013] Karaklajic, D., Schmidt, J-M., and Verbauwhede, I. Hardware Designer’s Guide to Fault Attacks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2013.
- [Nashimoto et al. 2017] Nashimoto, S. Homma, N., Hayashi, Y., Takahashi, J., Fuji, H., Taoki, T.: Buffer overflow attack with multiple fault injection and a proven countermeasure. J. Cryptographic Engineering 7(1): 35-46 (2017)
- [Verbauwhede et al., 2011] Verbauwhede, I., Karaklajic, D., and Schmidt, J.-M. (2011). The Fault Attack Jungle - A Classification Model to Guide You. In Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011), pages 3–8. IEEE.
- [Zussa et al., 2014] Zussa, L., Dehbaoui, A., Tobich, K., Dutertre, J-M., Maurine, P., Guillaume-Sage, L., Clediere, J., and Tria, A. Efficiency of a glitch detector against electromagnetic fault injection. In Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014.

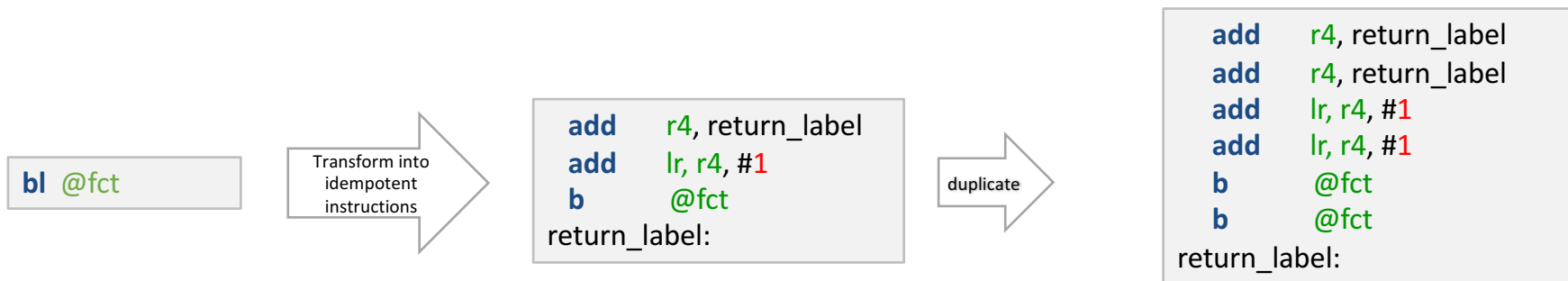
References


- [Akkar et al., 2003] Akkar, M.-L., Goubin, L., and Ly, O. (2003). Automatic Integration of Counter-Measures Against Fault Injection Attacks. In Proceedings of E-Smart'2003, pages 1–13, Nice.
- [Barengi et al. 2010] Barengi, A., Breveglieri, L., Koren, I., Pelosi, G., and Regazzoni, F. (2010). Countermeasures against fault attacks on software implemented AES. In Proceedings of the 5th Workshop on Embedded Systems Security (WESS'10), pages 1–10, ACM Press.
- [Barry et al., 2016] Barry, T., Couroussé, D., and Robisson, B. (2016). Compilation of a Countermeasure Against Instruction-Skip Fault Attacks. In Proceedings of the 3rd Workshop on Cryptography and Security in Computing Systems (CS2), pages 1–6, ACM Press.
- [De Keulenaer et al., 2016] De Keulenaer, R., Maebe, J., De Bosschere, K., and De Sutter, B. (2016). Link-time smart card code hardening. International Journal of Information Security, 15(2):111–130.
- [Goloubeva et al., 2005] Goloubeva, O., Rebaudengo, M., Sonza Reorda, M., and Violante, M. (2005). Improved software-based processor control-flow errors detection technique. In Annual Reliability and Maintainability Symposium, pages 583–589, IEEE.
- [Oh et al.] [Oh et al., 2002a] Oh, N., Shirvani, P. P., and McCluskey, E. J. (2002a). Control-flow checking by software signatures. IEEE Transactions on Reliability, 51(1):111–122.
- [Rauzy et al., 2015] Pablo Rauzy, Sylvain Guilley. A formal proof of countermeasures against fault injection attacks on CRT-RSA. J. Cryptographic Engineering 4(3): 173-185 (2014)
- [Reis et al., 2005] Reis, G., Chang, J., Vachharajani, N., Rangan, R., and August, D. (2005). SWIFT: Software Implemented Fault Tolerance. In Proceedings of the 2005 International Symposium on Code Generation and Optimization (CGO'05), pages 243–254. IEEE.
- [SIED, 2003] [Nicolescu et al., 2003] Nicolescu, B., Savaria, Y., and Velazco, R. (2003). SIED: Software Implemented Error Detection. In Proceedings of the 18th International Symposium on Defect and Fault Tolerance in VLSI Systems

Countermeasures for code integrity

Redundancy-based protections

- More complex transformation of non-idempotent instructions
- The function call example



 Moro et al. *Formal verification of a software countermeasure against instruction skip attacks.*
Journal of Cryptographic Engineering 2014.