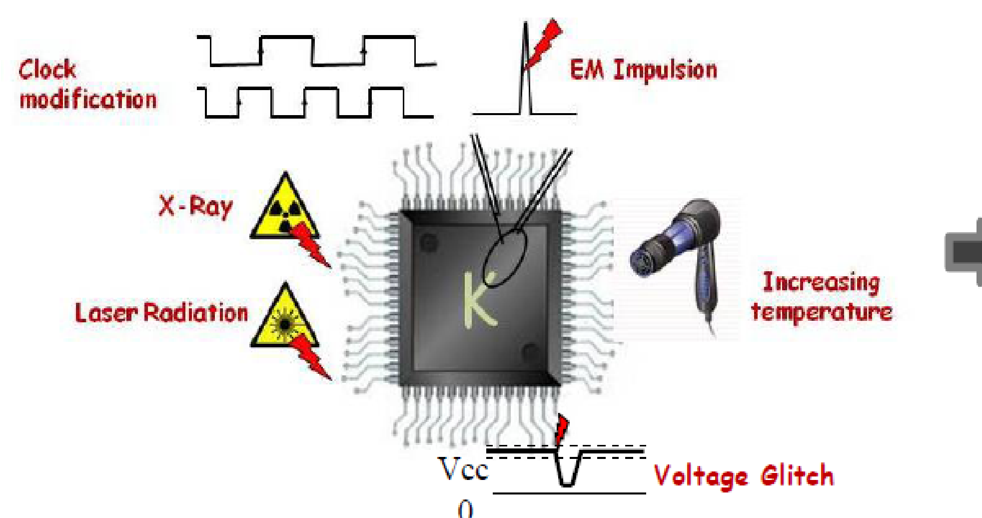
Two Fault Laser Attacks

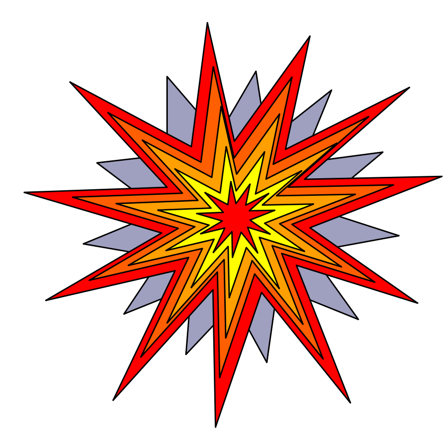
FDTC
Santa Barbara
August 21st 2010

Elena Trichina
Roman Korkikyan

Fault attacks

- Alter the expected behavior of a device by modifying:
  - Process flow
  - Data
- Introduced by Bellcore Labs in 1997
  - Theoretical attack on RSA-CRT
  - Allows to get the whole key (whatever the size) with a single faulty signature (whatever the fault)
  - Now widespread: all crypto algorithms are vulnerable
- Different ways to induce faults
  - Glitch on VCC, on clock, on whatever available input
  - Laser (with different wavelengths)
  - White light
  - Alpha particles
  - Electro magnetic emission
  - Temperature

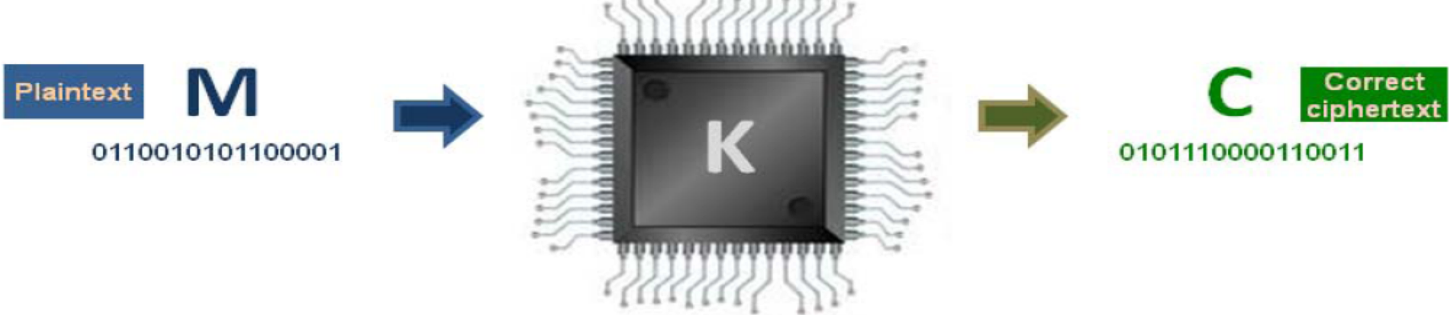




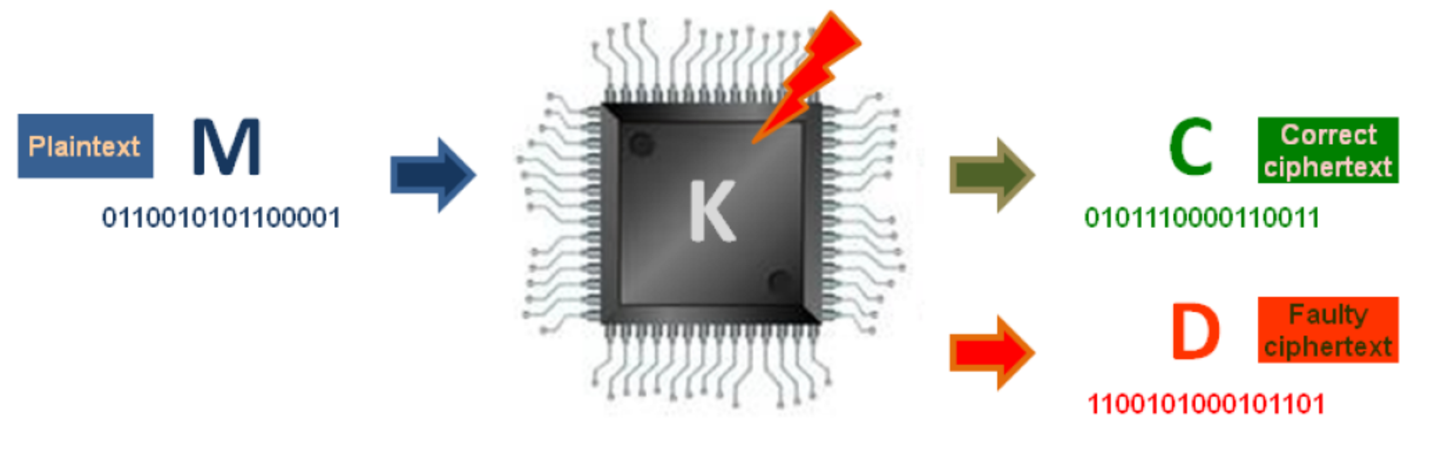
Fault attacks

- Not dedicated to crypto exclusively
  - Change value of registers/memory and escalate privileges
  - Force authentication without knowledge of the key
  - …
- When applied to crypto algorithms, fault injection rarely leads directly to key recovery
- A fault attack starts with an attack model
  - Clarify capabilities of the attacker
  - Specify types of errors, timing and location precision of the fault injection, the number of faults

Differential Fault Analysis (DFA)



- DFA exploits the differences between correct and faulty outputs of the cryptographic computations to discover the secret (e.g., a secret key)

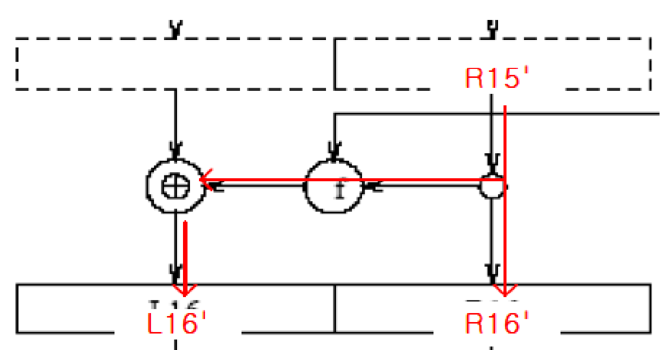


Fault attacks: examples

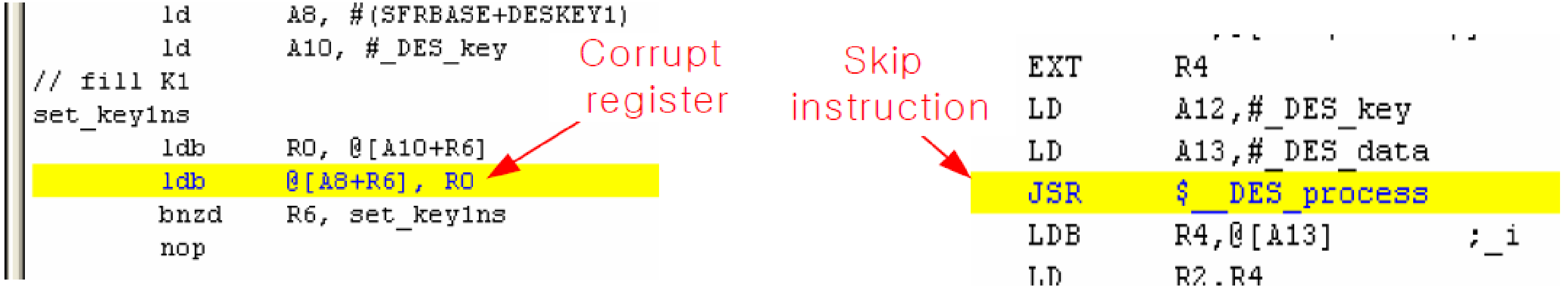
Bellcore attack on RSA CRT
(1996)

$$Sp = M^d \bmod P$$
$$Sq' = M^q \bmod Q$$
$$S' = CRT(Sp, Sq')$$
$$GCD(S - S', N) = p$$

DFA on DES

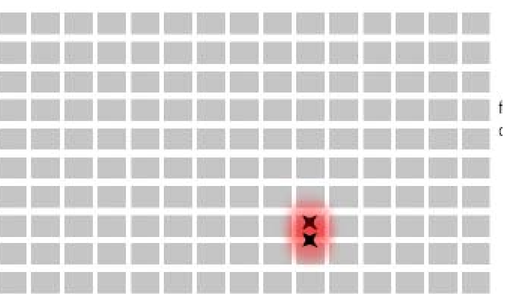


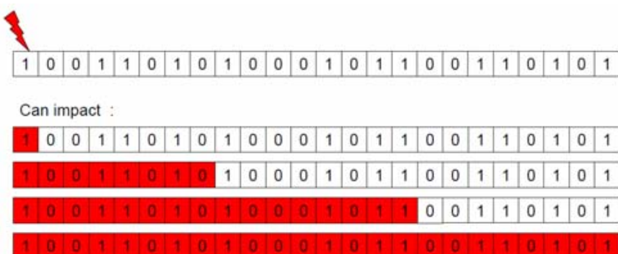
Fault attack on Operating System



Fault models

- Specification of a fault model includes various parameters:
  - Control on the fault location
    - No control
    - Loose control
    - Complete control
  - Control on the timing
    - No control
    - Loose control
    - Precise control
  - A number of bits affected
    - Single bit
    - Byte/half-word/word
    - Random number of bits
  - The fault type
    - Stuck at fault (stuck at one, stuck at zero)
    - Bit flip
    - Random fault
  - The effect of the fault
    - Transient, permanent or destructive





Generic countermeasure: duplication

- Execute (parts of) the algorithm twice and compare the results
  - Doubling :

Input

I want
to find
the secret key

Cryptographic operation

Cryptographic operation

=

Output

68sdLkjù£kj
df65g5KUE
4fos9uPrg9iè
6dskoj%F87
34dgfk9m°k6

  - Doubling :

Input

I want
to find
the secret key

Cryptographic operation

Cryptographic operation

=

Fault detected !

Generic countermeasure: verification

- "Invert" cryptographic computations:
  - Sign – Verify
  - Encrypt - Decrypt

– **Verification :**

Input

I want
to find
the secret key

Cryptographic operation

Inverse of the
cryptographic operation

=

Output

68sdLkjù£kj
df65g5KUE
4fos9uPrg9iè
6dskoj%F87
34dgfk9m°k6

– **Verification :**

Input

I want
to find
the secret key

Cryptographic operation

Inverse of the
cryptographic operation

=

Fault
detected !

Infective method

- Exists for public key algorithms
  - An algorithm is modified in such a way that, if the error is injected, it is used in some redundant computations which get “interwoven” into output values corrupting them to a degree when no analysis can be performed on faulty results

Second order fault model (2OFA)

- Generic modeling of a second order fault attack is difficult as any type of a single fault at instant T1 can be combined with any type of a single fault at instant T2

Parameter	Possible Values
location	no control, loose control, or complete control
timing	no control, loose control, or precise control
number of bits	single faulty bit, few faulty bits, or a random number of faulty bits
fault type	stuck-at fault, bit flip fault, random fault, or bit set or reset fault (to be defined in Section 1.3)
probability	(various possible values)
duration	destructive, permanent, or transient faults

Parameter	Possible Values
location	no control, loose control, or complete control
timing	no control, loose control, or precise control
number of bits	single faulty bit, few faulty bits, or a random number of faulty bits
fault type	stuck-at fault, bit flip fault, random fault, or bit set or reset fault (to be defined in Section 1.3)
probability	(various possible values)
duration	destructive, permanent, or transient faults

- Simplified second order fault model:
  - Because modification of the hardware used for a fault injection between two faults are difficult to modify, the type of the faults at T1 and T2 are similar
  - 2OFA

Danger of 2OFA

– Doubling :

– Verification :

2OFA model

- Doubling :

Input

I want
to find
the secret key

Cryptographic operation

Cryptographic operation

=

Faulty Output

6hi84dfs46mp
sdfUTYB4e
zs78dfs46dso
iu41jn5df

- Verification :

Input

I want
to find
the secret key

Cryptographic operation

Inverse of the
cryptographic operation

=

Faulty Output

84dfs46mptez
UTYB4efd
8dfs46dsofd
1jn5dfgfd

2OFA model

– Doubling :

Input

I want
to find
the secret key

Cryptographic operation

Cryptographic operation

=

Faulty Output

i8dfs46mp45
sIYB4e45
dfs46dso45
n5df472

– Verification :

Input

I want
to find
the secret key

Cryptographic operation

Inverse of the
cryptographic operation

=

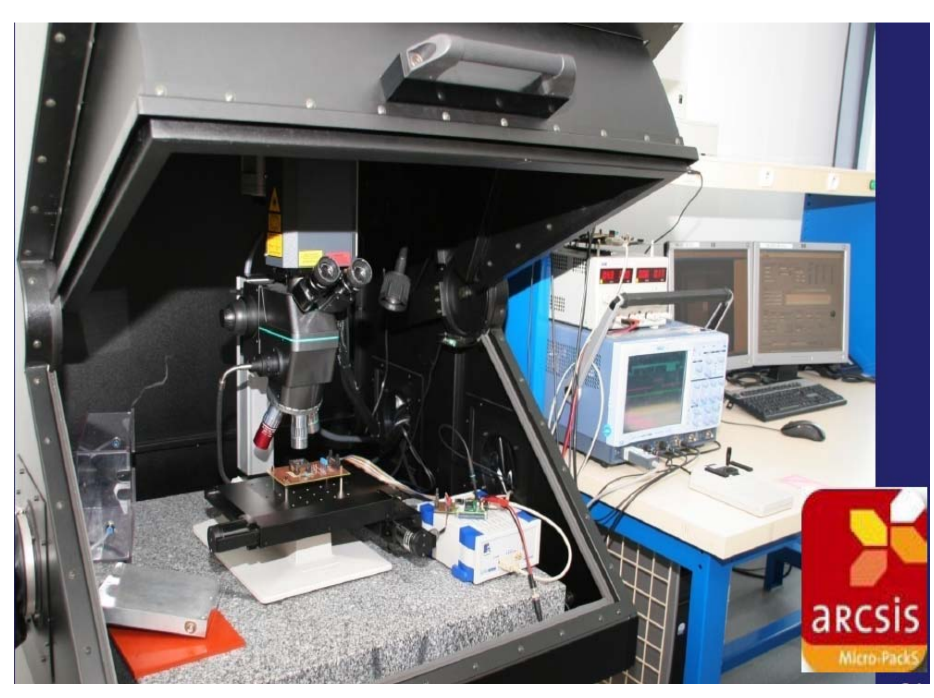
Fault
detected !

Are 2OFA practical?

- C. H. Kim and J.-J. Quisquater, “Fault Attacks on CRT based RSA: New Attacks, New Results, and New Countermeasures“, WISTP 2007, LNCS 4462
  - First practical glitch attack on CRT-RSA implemented on 8-bit microcontroller
- This talk is dedicated to practical aspects to 2OFA
- We present first successful laser 2OFA on a complex general-purpose 32 bit microcontroller based on ARM Cortex M3

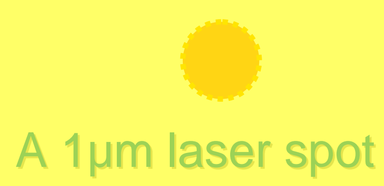
Why laser attacks?

- Good precision of fault injection:
  - Temporal (can target a particular instruction)
  - Spatial (can target a particular location: variable/byte/bit)
- Very efficient but require mastery of the bench and can be destructive
- Many parameters to manipulate and check
- Preparation of a chip is necessary



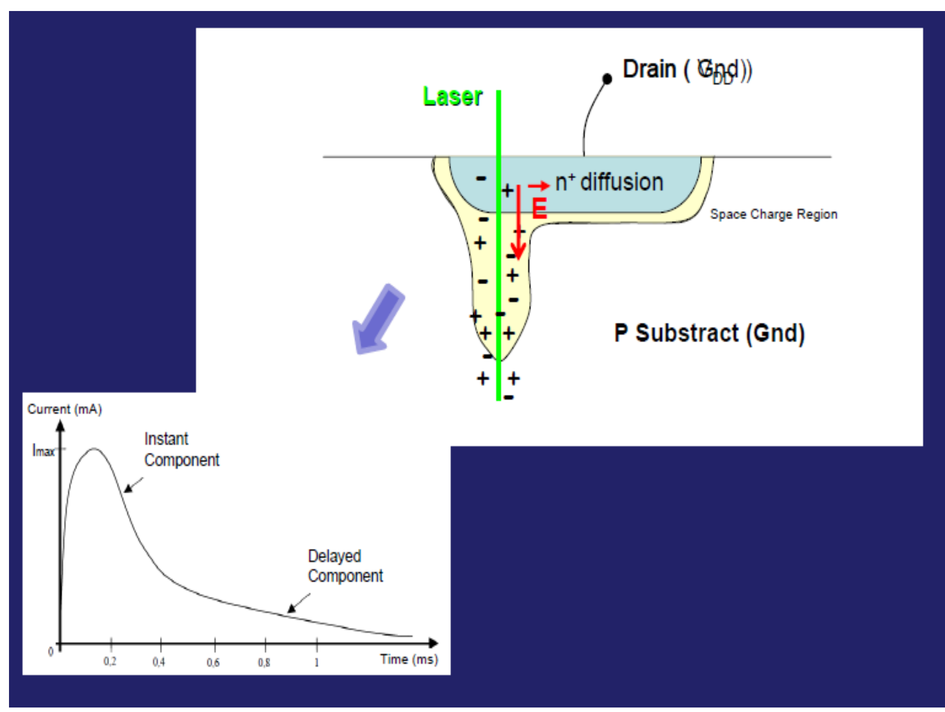
Technology size vs. laser spot size

	Transistor	SRAM Cell
350nm		
130nm		
90nm		
65nm		



A 10 µm laser spot

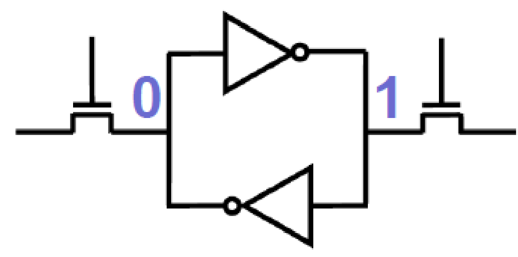
Laser effect on IC

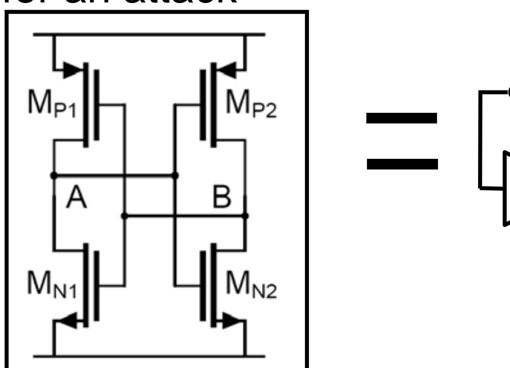


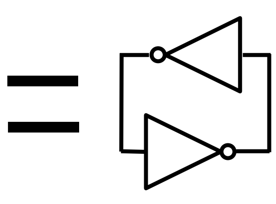
- Laser/silicon interaction is mainly photoelectric
- In silicon, the photoelectric effect is the absorbtion of a photon by an electric carrier to form electron-hole pair
  - Photoelectric interaction of a laser beam with silicon results in electron/hole pair generation on the path of the beam
  - The generated pairs can be separated by electrical fields in the device leading to different photocurrents
  - These transient currents may affect functionality of the transistors

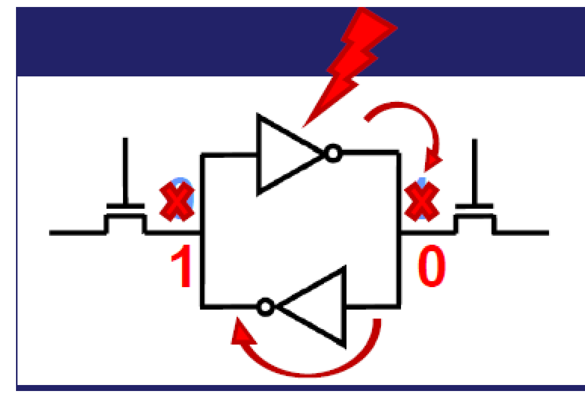
Laser effect on IC functionality

- On SRAM laser exposure is known to cause bit-flips
  - A one-bit SRAM cell is made of two cross-coupled inverters
  - The state of four transistors encode the stored value
  - Created by a laser/silicon interaction transient current inverts the output of one of the inverters
  - This voltage inversion is in turn applied to the second inverter switching it in an opposite state
  - A bit flip happens
- A phenomenon called Single Event Upset (SEU). Used for failure analysis. Can be used for an attack





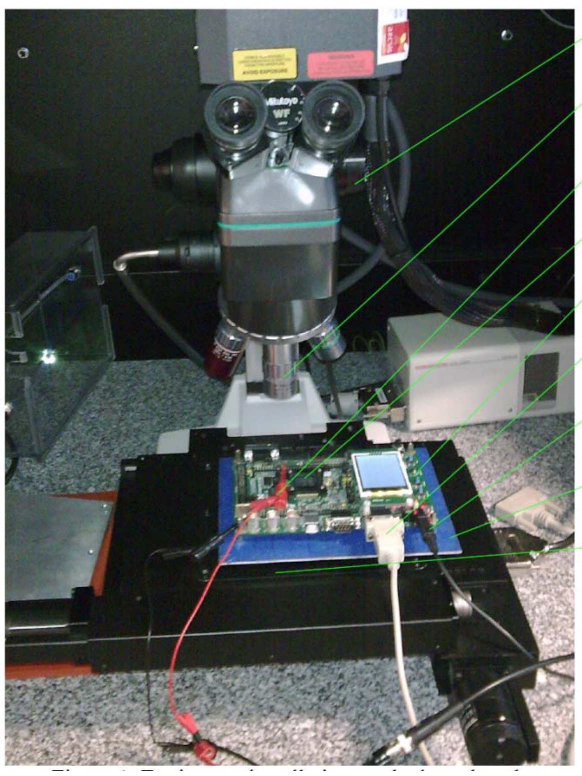




CMOS bistable: the basic memory element

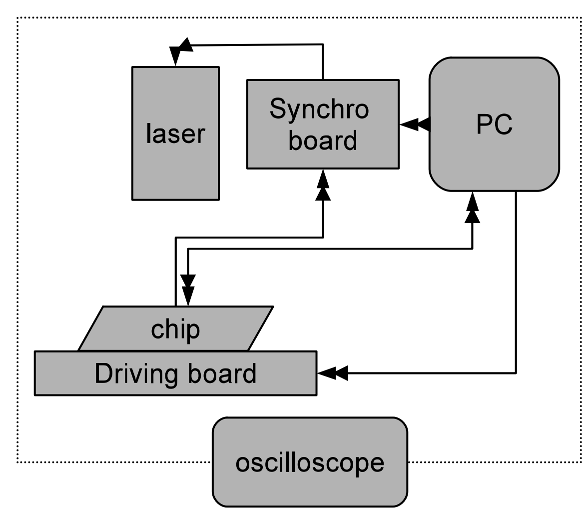
Laser

- YAG Pulse Laser
  - Two wavelengths (green 532nm wavelength, infrared 1064 nm wavelength)
  - Different lenses (GR: 2x, 20x; IR: 50x, 100x)
- 2500 um source Gaussian spot diameter
- X and Y apertures 35x35 µm tunable from 0% to 100%
- Energy level tunable from 0% to 100%
- Duration of a shot fixed at 5 nsec
- Adjustable timings of shots
- X-Y table to move the board; step 1 µm
- Microscope with camera to choose the hit area
- Focus: important parameter



Laser bench

- YAG pulse laser: shots when triggered
- X-Y table
  - Can be moved manually while selecting the start and the end position of the experiment
  - Moves the chip automatically during the experiment in accordance with selected parameters
- Oscilloscope: LeCroy 10GHz
  - Adjust and visualize the trigger from the board
  - Visualize the triggers from the bench
  - Visualize the power traces
  - It's possible to see the exact time of a shot
- Microscope with camera allows to choose the start and the end position of the experiment
- PC:
  - Runs a dedicated LabView interface
  - Allows a user to configure the Laser and the X-Y table
  - Allows the user to define the scenario of the experiment
  - Drives the equipment according to the settings
  - Sends commands to the board using a serial cable and saves the output to a file
- Synchroboard
  - Sets up a delay between the trigger signal and a shot with a step 10nsec



Laser bench software

- LabView interface helps automate an experiment

Play scenario without laser

Progress of experiment

Files with results

X-Y table displacement

Aperture

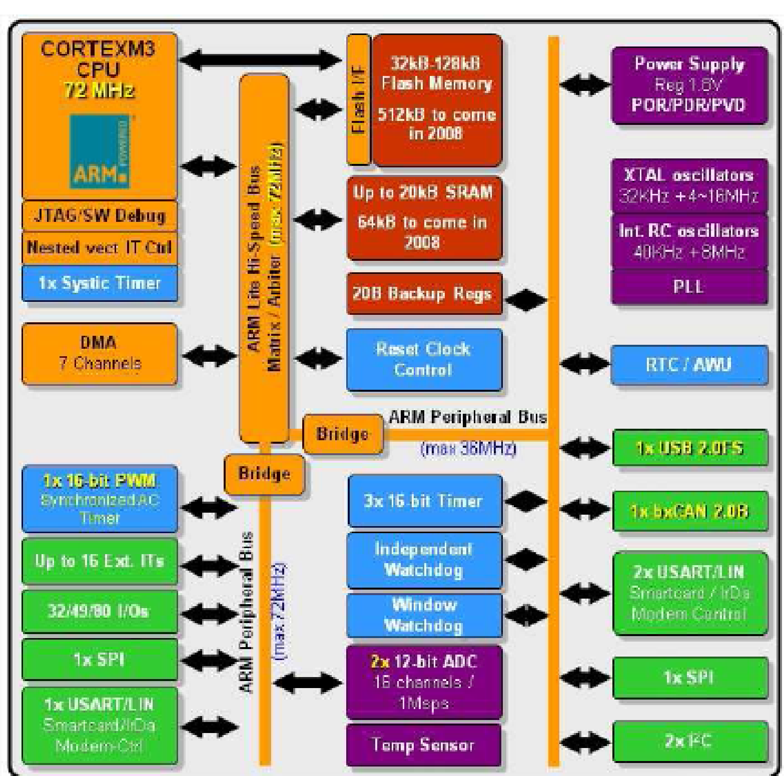
Timing/Delay

Wavelength

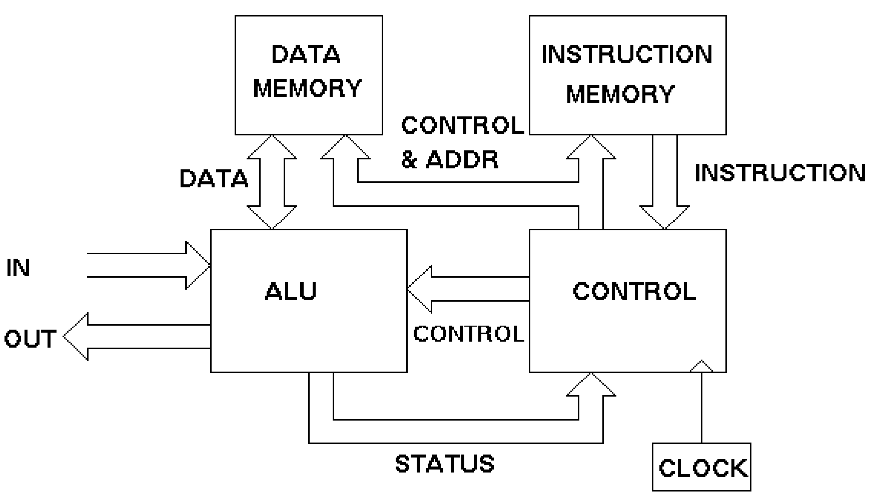
Energy

System on Chip

- 32-bit microcontroller based on ARM Cortex-M3 core
- Memory:
  - Embedded flash 512KB
  - Embedded RAM up to 64KB
- Code executed from flash
- Many peripherals, both analogue and digital
- Technology:
  - 130 nm
  - 6 metal layers
- Many safety and security features:
  - programmable voltage detector, embedded voltage regulator, internal clocks, clock detector, tamper bit, exception fault handling, watchdogs, emergency stop, write once registers, backup register, flash memory protection



ARM architecture



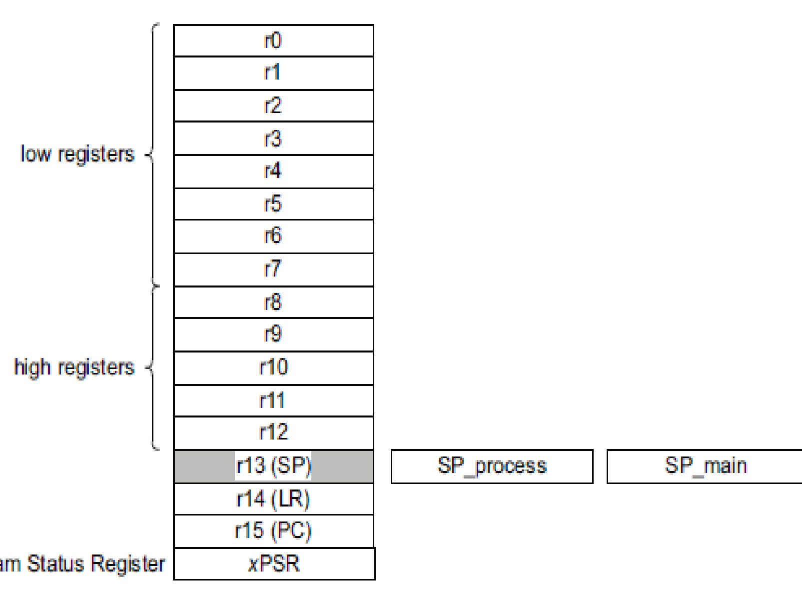


Figure 2-1 Processor register set

What can be done with SEU?

- ARM is a load/store machine
- Based on registers
- Change a register bit:
  - ➔ change control flow
  - ➔ change address
  - ➔ change operation
  -

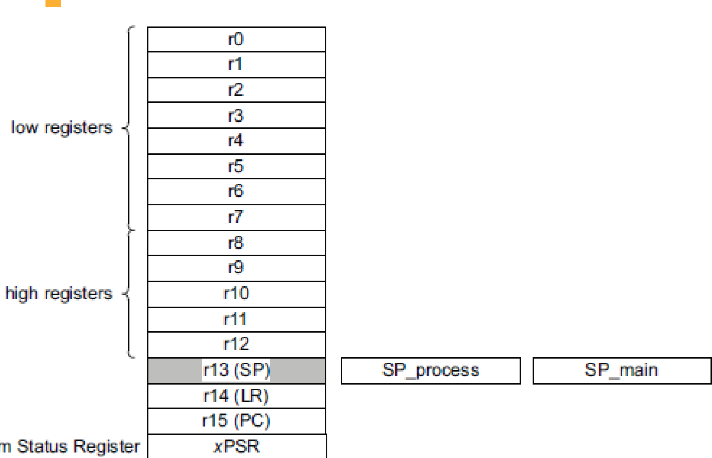


Figure 2-1 Processor register set

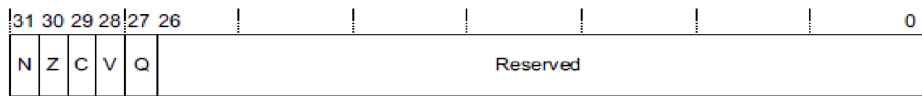


Figure 2-2 Application Program Status Register bit assignments

Table 2-1 describes the bit assignments of the APSR.

Table 2-1 Application Program Status Register bit assignments

Field	Name	Definition
[31]	N	Negative or less than flag: 1 = result negative or less than 0 = result positive or greater than.
[30]	Z	Zero flag: 1 = result of 0 0 = nonzero result.
[29]	C	Carry/borrow flag: 1 = carry or borrow 0 = no carry or borrow.
[28]	V	Overflow flag: 1 = overflow 0 = no overflow.

Data Processing and Shift Instruction

31 30 29 28	27 26 25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6 5	4	3 2 1 0
Condition Field	0 0 0	Operation Code	S	Source Register 1	Destination Register	Shift Register	0	Shift	1	Source Register 2

(a)

Branch and Branch-Link Instruction

31 30 29 28	27 26 25	24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Condition Field	1 0 1	L	24 bit Offset

(b)

Protected CRT-RSA: verification method

Algorithm 1 Protected CRT-RSA	
Inputs: CRT-RSA private key and message C **Outputs**: RSA signature or error detection	
Mathematical expression	**Description**
1. $M_p = C^{d \bmod p-1} \pmod p$ 2. $M_q = C^{d \bmod q-1} \pmod q$ 3. $M = \left((M_p - M_q) \cdot i_q \bmod p\right) q + M_q$	Signing message C with a CRT-RSA private key
4. $e_p = d_p^{-1} \bmod (p-1)$ 5. $e_q = d_q^{-1} \bmod (q-1)$ 6. $C_p = M^{e_q} \bmod p$ 7. $C_q = M^{e_p} \bmod q$ 8. $C' = \left((C_p - C_q) \cdot i_q \cdot \bmod p\right) q + C_q$	Verification of the signature obtained in Step 1. Here one does not require the knowledge of e a priori, instead available input parameters are used to recover plaintext.
9. *if* $C' == C$ *then*	Comparison
10. *return* M	Correct answer if there is no fault
11. *else* 12. *return* *ErrorMessage*	Error message in the case of fault

- A. Boscher, H. Handschuh, E. Trichina, Chinese Remaindering in Both Directions, 2010 http://eprint.iacr.org
- Efficient signature verification-based countermeasure
- All computations on half-sized data

2OFA model: by-pass verification

Algorithm 1 Protected CRT-RSA	
Inputs: CRT-RSA private key and message C **Outputs**: RSA signature or error detection	
Mathematical expression	**Description**
1. $M_p = C^{d \bmod p-1} \pmod p$ 2. $M_q = C^{d \bmod q-1} \pmod q$ 3. $M = \left((M_p - M_q) \cdot i_q \bmod p\right) q + M_q$	Signing message C with a CRT-RSA private key
4. $e_p = d_p^{-1} \bmod (p-1)$ 5. $e_q = d_q^{-1} \bmod (q-1)$ 6. $C_p = M^{e_q} \bmod p$ 7. $C_q = M^{e_p} \bmod q$ 8. $C' = \left((C_p - C_q) \cdot i_q \cdot \bmod p\right) q + C_q$	Verification of the signature obtained in Step 1. Here one does not require the knowledge of e a priori, instead available input parameters are used to recover plaintext.
9. *if* $C' == C$ *then*	Comparison
10. *return* M	Correct answer if there is no fault
11. *else* 12. *return* *ErrorMessage*	Error message in the case of fault

- First fault injected during exponentiation
- Second fault by-passes the countermeasure

Infective method [BHT]

Algorithm 2 Second Order resistant CRT-RSA with infective method	
Inputs: CRT RSA private key and message C **Outputs**: RSA signature or error detection	
Steps 1-8 are the same as in Algorithm 1	
9. *if* $C{=}{=}C$ *then*	Comparison.
10. *return* $M + C_p - (C \bmod p) + C_q - (C \bmod q)$	Infect the answer with redundant information which spreads an error if Step 9 is skipped
11. *else* 12. *return* *ErrorMessage*	Error message in the case of fault in Step 9

- Confirms a well-known wisdom that sometimes a countermeasure against new attack creates a vulnerability wrt. the old one

Single fault attack on infective method

Algorithm 2 Second Order resistant CRT-RSA with infective method	
Inputs: CRT RSA private key and message C **Outputs**: RSA signature or error detection	
Steps 1-8 are the same as in Algorithm 1	
9. *if* $C == C$ *then*	Comparison.
10. *return* $M + C_p - (C \bmod p) + C_q - (C \bmod q)$	Infect the answer with redundant information which spreads an error if Step 9 is skipped
11. *else* 12. *return* *ErrorMessage*	Error message in the case of fault in Step 9

$$R' = M - (C \bmod p) + C_q - (C \bmod q)$$

$$R' - R = C \bmod p$$

$$C = (R' - R) + k \cdot p$$

$$C - (R' - R) = k \cdot p$$

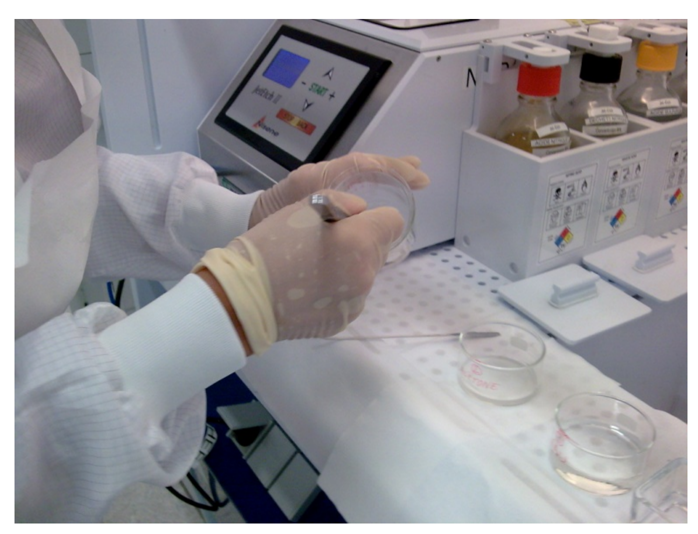
$$GCD\left(C - (R' - R), N \right) = GCD\left(k \cdot p, p \cdot q \right) = p$$

From 2OFA theory to 2OFA practice

- 2OFA are possible in theory and may break many countermeasures
- Yet… why there were not but ONE publication on their practical implementation ?
  - Papers were rejected by PC?
  - There were no practical implementations?
- And certainly no two fault laser attacks were ever published

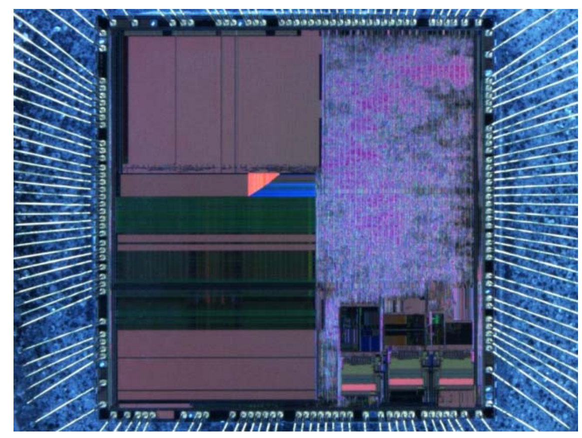
Preparation steps

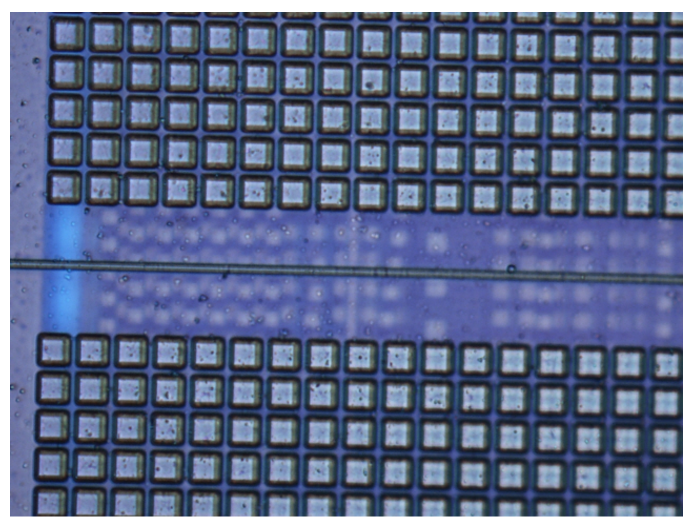
- To perform front-side laser attacks, a decapsulated chip is needed
- For de-capsulation we used chemical etching
- Performed using JetEtch II tool which, after all the parameters (e. g., type of acid, temperature, time, etc.) are set, runs the process to the completion automatically
  - Selection of parameters is “know-how”



Decapsulated SoC

First setbacks



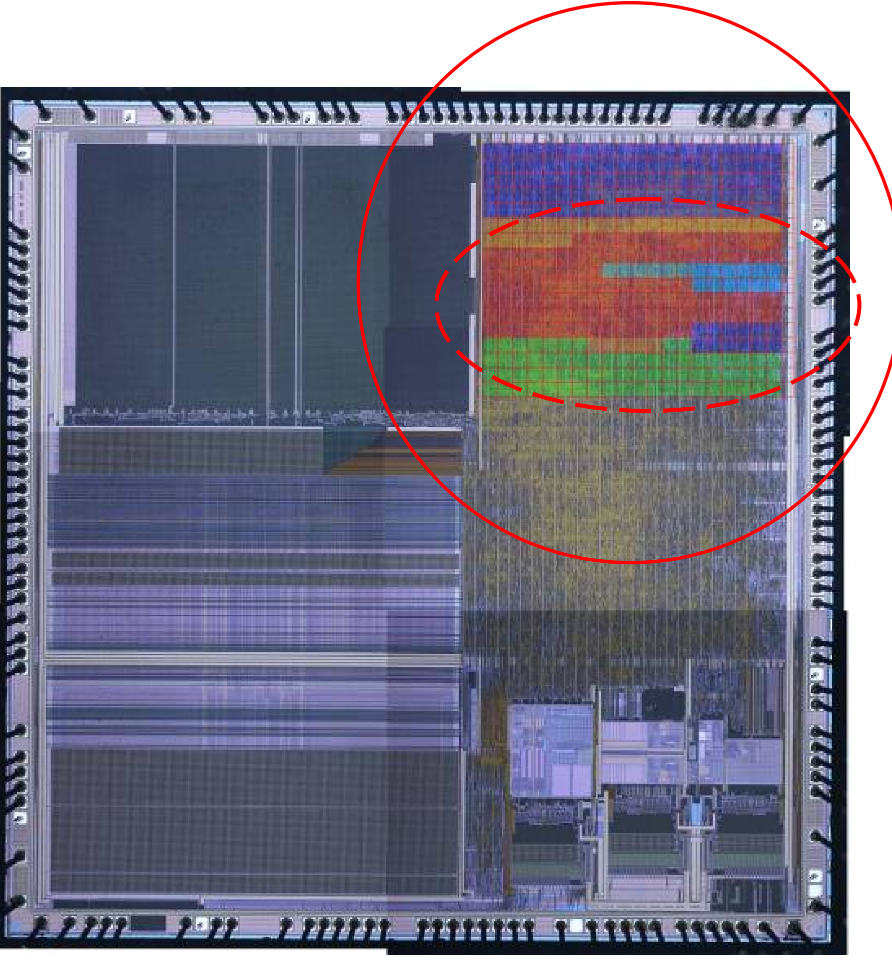


- Digital components are implemented in glue logic
- CPU occupies only 20% of the logic area

- SRAM is covered with metal tiles (1 layer)
- Flash is covered with metal tiles (2 layers)

Where is the CPU?

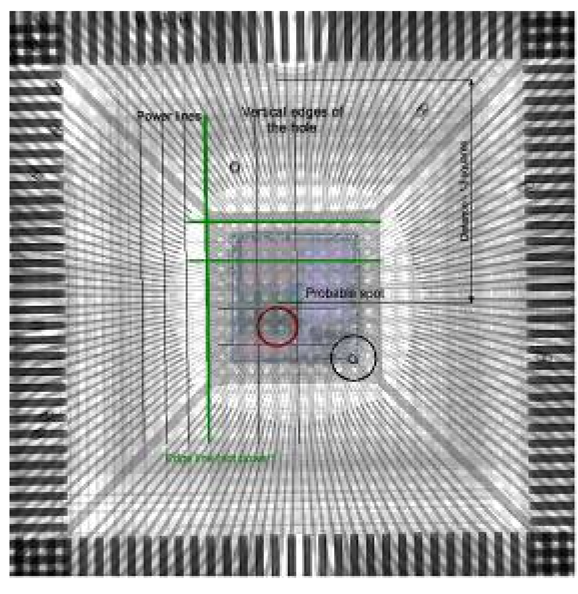
- Scan with EM probe helps to visualize active areas



Algorithm for running a laser bench

2. The PC sends a command (e. g. start execution of cryptographic algorithm) to the chip.
3. The chip starts execution and at some moment of time it raises up a trigger.
4. The trigger is recognized by the synchroboard.
5. The synchroboard generates another trigger for the laser with a predefined delay.
6. The laser receives the trigger and after the delay it shoots.
7. The chip prints the result to a serial port.
8. This result is recorded by the computer.
9. The computer maintains the current state of the experiment, i. e., it moves the driving board, changes the delay of the synchroboard if necessary and does other routines specified in the attack scenario.
10. Everything returns to the step 1.
11. Steps 1-10 continue to run until the driving board reaches the end point.

Additional model parameter: sheer luck



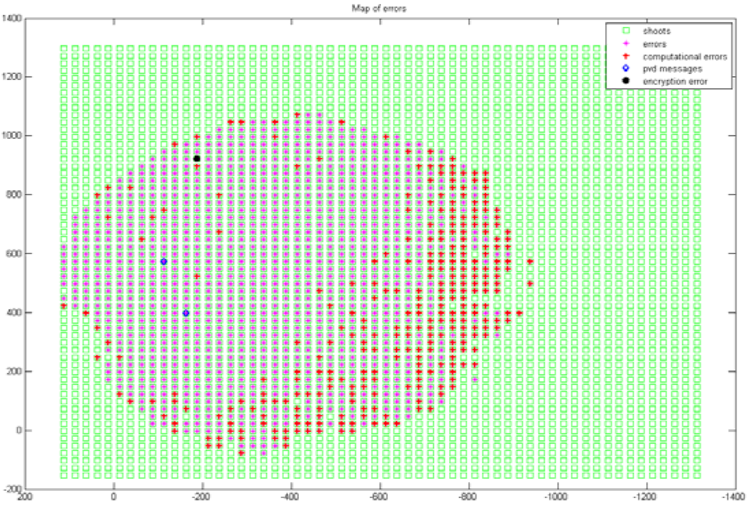


Figure 2. Approximate errors location for the front side

Not glamorous attack routine

- De-package a chip
- Prepare SW for an attack:
  - for driving a bench, for communication with the bench, for investigation of possible fault models and for post-processing of faulty results
- Find vulnerable spot on a chip
  - by repeatedly scanning a chip with a laser (shooting) varying laser parameters while the chip runs programs
- Collect and analyze the results; infer fault types,…
- Refine an attack, refine timing of shots, try different laser bench parameters…
- Until the algorithm is broken

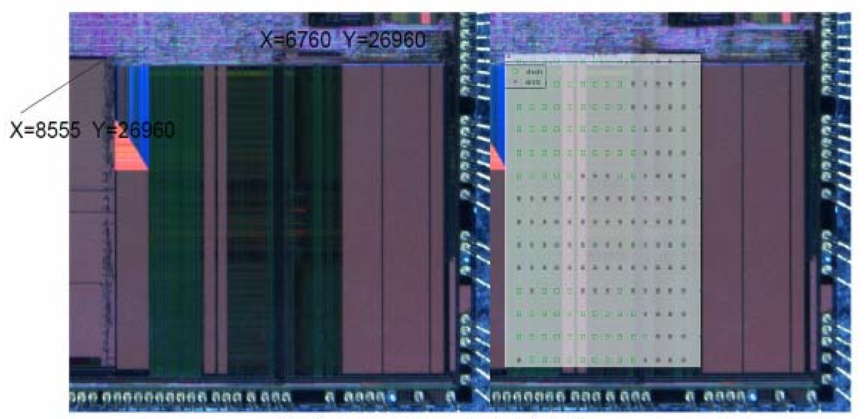
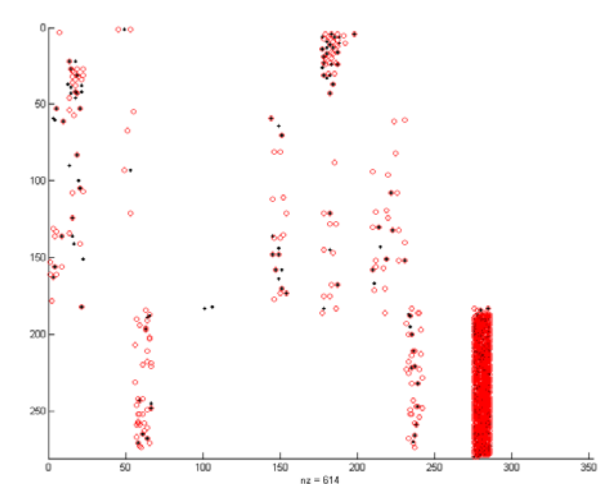
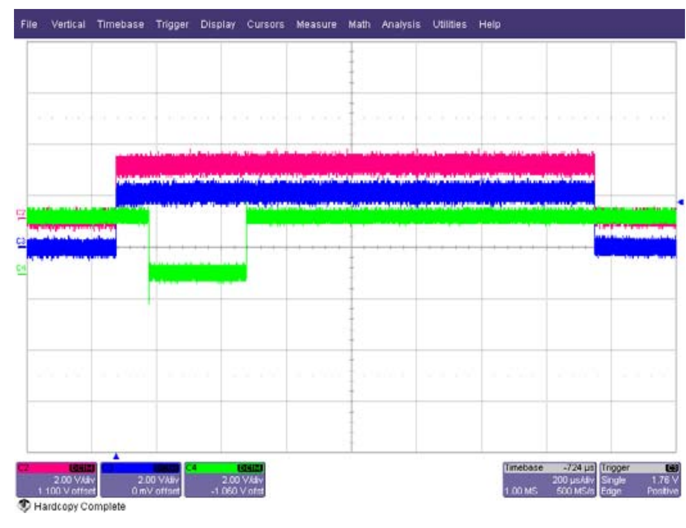
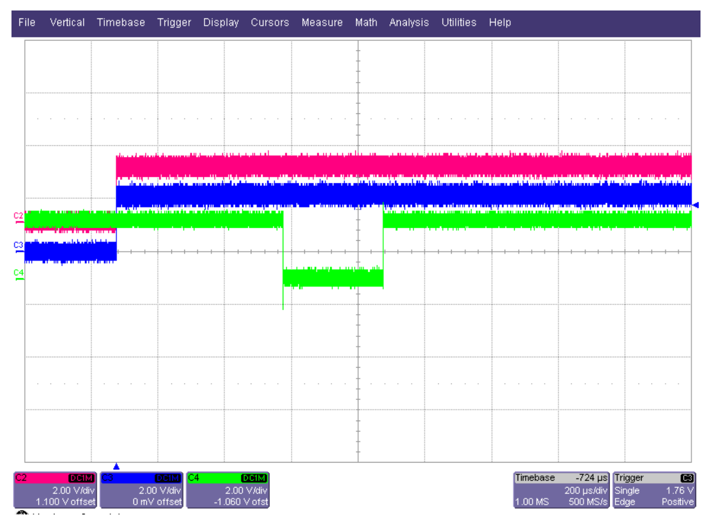


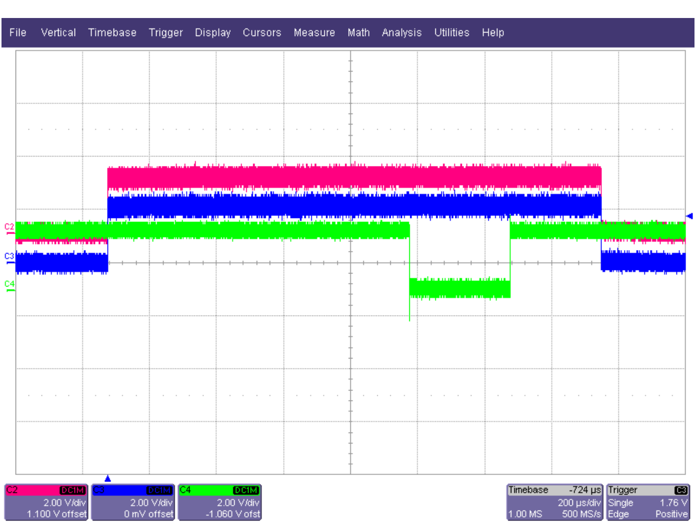
Figure 2. Approximate errors location for the front side

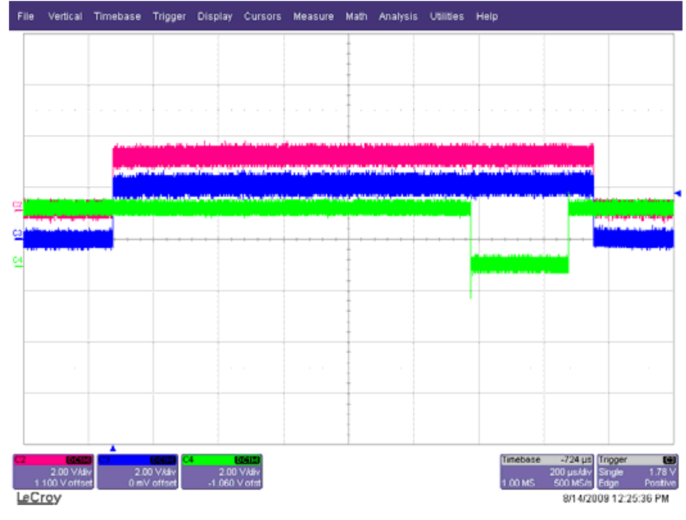


Finding precise time for a shot









Systematic approach

- Record laser parameters (location, energy level, focused vs. unfocused,…)
- Finding a proper time of a shot is the most important factor for obtaining exploitable errors
  - Time of the shot is adjustable by changing trigger parameters
- All faulty results are recorded in a separate file while running an experiment
- Run post-processing on faulty results ➔ recover potential key ➔ check

5	Single byte error in first subkey in round 10	Green Aperture (%): X = 50,0 Y = 50,0 Energy: 0,6%	
		(7350,25800,0.44) (7250,25500,0.44) (7150,25500,0.44) (7050,25500,0.44) (6950,25500,0.44) (6950,25300,0.44)	a4fb89db e9cc53cf 921109c1 ca50c573 7efb89db 33cc53cf 481109c1 1050c573 DA000000 DA000000 DA000000 DA000000
		Green Aperture (%): X = 50,0 Y = 50,0 Energy: 0,5%	
		(7450,26600,0.44)	a4fb89db e9cc53cf 921109c1 ca50c573 a4fb8957 e9cc5343 9211094d ca50c5ff 0000008C 0000008C 0000008C 0000008C
		(7150,26600,0.44) (7050,26600,0.44)	a4fb89db e9cc53cf 921109c1 ca50c573 a4fb89b3 e9cc53a7 921109a9 ca50c51b 00000068 00000068 00000068 00000068
		(7350,25600,0.44)	a4fb89db e9cc53cf 921109c1 ca50c573 a4fb895e e9cc534a 92110944 ca50c5f6 00000085 00000085 00000085 00000085
		(7250,25500,0.44)	a4fb89db e9cc53cf 921109c1 ca50c573 8ffb89db c2cc53cf b91109c1 e150c573 2B000000 2B000000 2B000000 2B000000

Skipping comparison

Code: if-condition instruction in terms of assembler code

```
0x08001160 A9FD      ADD      r1,sp,#0x3F4
0x08001162 A82B      ADD      r0,sp,#0xAC
0x08001164 F000F881  BL.W     CompareBig (0x0800126A)
   509:                 if (i == 0)     {
   510:                  /* write to vector of char */
0x08001168 B930      CBNZ   r0,0x08001178    ⍰ Check
   511:
W32_to_W8(m.w,P_pOutput,P_pPrivCRTKey->modulus_size);
   512:                 } else {
0x0800116A 4639      MOV      r1,r7
0x0800116C F1060008  ADD      r0,r6,#0x08
0x08001170 6822      LDR      r2,[r4,#0x00]
0x08001172 F000FCF8  BL.W     W32_to_W8 (0x08001B66)
0x08001176 E003      B        0x08001180
   513:                           printf("Fault %d\n",i);
   514:                 }
   515:             }
0x08001178 4601      MOV      r1,r0
0x0800117A A004      ADR      r0,{pc}+2  ; @0x0800118C
0x0800117C F002FC24  BL.W     __1printf (0x080039C8)
```

- Compare and Branch on Non Zero
  - CBNZ Rn, label
- Register R0 keeps the result of comparison between two large numbers, initial and re-computed plaintexts
- If they are equal the return value is 0 otherwise the result can be -2,-1, 1 or 2 depending on some conditions
- CBNZ command does not change the conditional flag in a program status register xPSR
- Several ways to skip /alter verification result:
  - Skip execution of CompareBig ➔ register R0 may not have a useful value
  - Force register R to 0
  - Skip CBNZ instruction ➔ the system increments PC and goes to “IF-YES” branch

Finding right timing for 2nd shot

File Vertical Timebase Trigger Display Cursors Measure Math Analysis Utilities Help

C4
C1
C3

C1 DC1M 1.00 V/div -2.000 V ofst

C3 DC1M 2.00 V/div -6.000 V ofst

C4 DC1M 200 mV/div 400.0 mV ofst

Timebase 0 ns 100 ns/div 10.0 kS 10 GS/s

Trigger C3 Normal 1.16 V Edge Negative

LeCroy

Waiting for Trigger

Two shots and two faults!

LeCroy

C1 DC1M
500 mV/div
1.000 V offset

C2 DC1M
2.00 V/div
-150 mV ofst

C4 DC1M
2.00 V/div
-4.190 V ofst

Timebase 17.6 ms
10.0 ms/div
1.00 MS 10 MS/s

Trigger C2
Normal 1.00 V
Width Positive

Private key recovery

Table 9. Examples of two fault attacks against CRT-RSA, based on a conditional check

Correct data	Faulty encryptions
Correctly encrypted message 2A 28 E0 F1 81 84 53 EF 29 CA D7 C2 43 BB BA 88 E3 CD 42 24 23 FA F1 BC 81 91 98 3C FE 2C 4B DD 23 8C 83 2B 8D 7D 7F B0 0F EE 06 1F 09 76 94 8B DC 6F BB DA 6A 45 FC 6F 47 32 9D 9D EA 0C E7 64 Modulus 3E D6 FC AA D0 61 4D 3C EA D5 1D 4B 81 A7 A7 54 BF B3 A2 9A CC 0D F3 28 51 7C C5 05 84 9F 0C C9 CF 88 42 3B 0B FD 67 F0 7B BA 53 60 64 06 DB 99 C8 DD 13 9B 3A 12 06 36 2A E4 5D 20 A5 75 EE AF The prime number, recovered from faulty encryptions: 67 FD 7D 60 C8 80 9D 78 09 9A FA D0 A0 B4 FC 07 68 53 5D D2 64 69 1C 5E 2A 00 AD AC A8 CA 19 0F	Faulty encrypted message 1 3A F4 F1 81 81 F5 08 9B 62 4A 12 71 B1 E9 9E 38 EC FC 9F 95 2F 5C D4 13 51 14 C5 3C 75 2C 74 87 2A 69 BE 48 72 AE A1 EA C7 5D B7 A1 B0 9B 92 37 A5 C3 7A AB 9D AC 5B 28 AE 74 CD 55 03 F2 04 F6 Faulty encrypted message 2 0A F3 1B B6 98 AC 38 98 2B 5B CE 3F 92 93 6F B3 75 CC 6F DE 55 B3 B1 3F 2C 1F 9A 5C 54 D1 0A AB AC 31 3B AE 63 FE 89 FB D7 22 A8 8A BC AF 0E 3E FD FD D1 9C 09 2E 69 65 67 20 15 1D 2D 1A 33 C0
Correctly encrypted message 3B A9 FC A4 8B F3 8C AF 25 AC 01 39 85 36 ED D0 29 48 2B 3A 6E E5 FB FD FA 11 80 64 1C 32 13 71 4C 45 97 E2 67 1D 2F 60 32 06 44 D3 34 87 80 CE 1B 8F 42 37 20 6E 3B 6E D0 20 73 48 D5 8B 22 3C Modulus 51 68 A0 CC 86 A1 38 90 71 E8 83 44 C2 87 F0 67 D9 A5 10 40 0C B7 5D 3D 47 B3 4C FA EE F0 97 60 F6 36 25 F4 78 DD 39 AD 7C E0 64 CD 3F EC EE DB 0A B7 22 FF E6 35 AA 18 E0 23 B6 A8 E9 2B 72 7D The prime number, recovered from faulty ciphertexts: 5F 6A 86 6C 9D 83 A6 B4 C7 43 2E 37 5C 25 92 43 20 AF BA AF EE 71 12 C0 F0 E0 7B FD 5F 7E 46 6B	Faulty encrypted message 1 13 06 E5 8A D6 FD 8A 7D B9 0D 5C 2F 99 46 02 4F 60 4C F3 B5 CE 6B DC 19 3E CC 5F A8 4D 86 8E 9E FE 5F 1C 34 48 B1 F9 D2 AF 4F 1E B9 11 79 9A 16 6A A7 1E 1A CB 00 65 F1 3C F6 D7 67 65 5D 75 7E Faulty encrypted message 2 4F 81 03 FE 28 8A A9 B4 BB 71 DF 4D 56 4F 07 5A BF A1 B3 EE 7B DE 71 B3 42 50 35 13 9A BE 2A AB FF EA 1D 6C 93 79 FE 94 62 36 B2 DF A5 36 42 8C 29 58 C6 F3 CC A8 57 B2 6A 0F 03 74 45 53 16 67 Faulty encrypted message 2 1F D9 EA E2 07 A5 C4 75 24 F6 87 94 5B CB 11 7F 25 7E 8E 15 8D 4E 8C 09 04 BA D1 CE CC AA 3F 5E FC 70 D7 D2 7C DE 36 25 FF 95 64 CF 6A E7 DA 30 55 F0 E7 4A 16 6F C4 2B 79 93 3E B2 1D 3A B3 5B

- Run Bellcore-style attack routine on every faulty result
- CRT-RSA (signature) is broken in one day
- Protected against single fault CRT RSA broken with two-fault attack in one week ➔
- New powerful practical attack ➔ Need new countermeasures
- Catalogue “useful” faults and develop new attack models & countermeasures

Skipping function calls: details

File Vertical Timebase Trigger Display Cursors Measure Math Analysis Utilities Help

C4
C1
C3

C1 DC1M	C3 DC1M	C4 DC1M	Timebase 0 ns	Trigger C3
1.00 V/div	2.00 V/div	200 mV/div	100 ns/div	Normal 1.16 V
-2.000 V ofst	-6.000 V ofst	400.0 mV ofst	10.0 kS 10 GS/s	Edge Negative

LeCroy

Waiting for Trigger

Analysis of unusual behavior

asm code for GPIO_ResetBits

```
352: void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
353: {
354:   /* Check the parameters */
355:   assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
356:   assert_param(IS_GPIO_PIN(GPIO_Pin));
357:
0x08003068 4770      BX       lr
358:   GPIOx->BRR = GPIO_Pin;
0x0800306A 6141      STR      r1,[r0,#0x14]
359: }
360:
361: /*******************************************************************************
362: * Function Name  : GPIO_WriteBit
363: * Description    : Sets or clears the selected data port bit.
364: * Input          : - GPIOx: where x can be (A..G) to select the GPIO peripheral.
365: *                  - GPIO_Pin: specifies the port bit to be written.
366: *                    This parameter can be one of GPIO_Pin_x where x can be (0..15).
367: *                  - BitVal: specifies the value to be written to the selected bit.
368: *                    This parameter can be one of the BitAction enum values:
369: *                       - Bit_RESET: to clear the port pin
370: *                       - Bit_SET: to set the port pin
371: * Output         : None
372: * Return         : None
373: *******************************************************************************/
374: void GPIO_WriteBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin, BitAction BitVal)
375: {
376:   /* Check the parameters */
377:   assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
378:   assert_param(IS_GET_GPIO_PIN(GPIO_Pin));
379:   assert_param(IS_GPIO_BIT_ACTION(BitVal));
380:
0x0800306C 4770      BX       lr
381:   if (BitVal != Bit_RESET)
382:   {
383:     GPIOx->BSRR = GPIO_Pin;
384:   }
385:   else
386:   {
0x0800306E 2A00      CMP      r2,#0x00
387:     GPIOx->BRR = GPIO_Pin;
388:   }
0x08003070 BF0C      ITE      EQ
0x08003072 6141      STREQ    r1,[r0,#0x14]
383:     GPIOx->BSRR = GPIO_Pin;
384:   }
385:   else
386:   {
387:     GPIOx->BRR = GPIO_Pin;
388:   }
0x08003074 6101      STRNE    r1,[r0,#0x10]
389: }
390:
391: /*******************************************************************************
392: * Function Name  : GPIO_Write
393: * Description    : Writes data to the specified GPIO data port.
394: * Input          : - GPIOx: where x can be (A..G) to select the GPIO peripheral.
395: *                  - PortVal: specifies the value to be written to the port output
396: *                    data register.
397: * Output         : None
398: * Return         : None
399: *******************************************************************************/
400: void GPIO_Write(GPIO_TypeDef* GPIOx, u16 PortVal)
401: {
402:   /* Check the parameters */
403:   assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
404:
0x08003076 4770      BX       lr
405:   GPIOx->ODR = PortVal;
0x08003078 60C1      STR      r1,[r0,#0x0C]
406: }
407:
408: /*******************************************************************************
409: * Function Name  : GPIO_PinLockConfig
```

asm code for GPIO_WriteBits

Laser shot

Raising trigger

Back to the main flow

It was compiled in such a way that there is a GPIO_WriteBit function exactly after the GPIO_ResetBit. It means, that if the return statement is missed, then the program enters to the GPIO_WriteBit and it starts to execute its instructions, until the new 'bx lr' is reached. In one point there is a function (line 0x08003074) which raises up the trigger again and it is observed on the oscilloscope.

Attack against infective method

```
i = CompareBig(&temp,&c);

if (i == 0)
{
        /*temp - is a initial plaintext
        mp  and  mq  -  parts  of  the  RSA  recombination
for the plaintext*/

        GPIO_ResetBits(GPIO_SMART, GPIO_Pin_11);

        /*m = m + mp*/
        AddBig(&m,&mp,&m);       □ skip instruction

        /*m = m + mq = m + mp + mq*/
        AddBig(&m,&mq,&m);

        /*result = c mod p*/
        ModularReduction(&c,&result,&p);

        /*temp = m - result = m + mp + mq - cp*/
        SubBig(&m,&result,&temp);

        /*result = c mod q*/


        ModularReduction(&c,&result,&q);

        /*m = temp - result = m + mp + mq - cp - cq*/
        SubBig(&temp,&result,&m);

        W32_to_W8(m.w,P_pOutput,P_pPrivCRTKey-
>modulus_size);
}else {
        printf("Fault %d\n",i);
   }
```

- After “subroutine skipping” had been mastered, an attack was easy and reproducible
- Because the countermeasure infects only half bytes of the result, it is easy to understand that a required error was injected

Table 10: The result of attack against basic infective method

Correct data	Faulty ciphertext
Correct plaintext 3A 7A 11 F7 04 FE A5 29 D6 06 6F 35 4D 7E 50 8F 51 3A 71 51 FA 7C 97 A9 63 74 04 03 24 97 9C E1 53 F6 53 35 AB 63 74 04 03 24 97 EE DF FF 18 A5 12 3B 8B 42 6B 74 9A 4B 20 8D 0D 18 4E 7A F8 B0	Incorrect ciphertext 3B A9 FC A4 8B F3 8C AF 25 AC 01 39 85 36 ED D0 29 48 2B 3A 6E E5 FB FD FA 11 80 64 1C 32 13 71 46 95 73 F4 2F 71 B0 2E 5A 8C 4F F4 F2 2B EF D8 20 BE 8C B5 AE 55 57 F1 DB D7 D2 0E 93 9C C7 D7
Correct ciphertext 3B A9 FC A4 8B F3 8C AF 25 AC 01 39 85 36 ED D0 29 48 2B 3A 6E E5 FB FD FA 11 80 64 1C 32 13 71 4C 45 97 E2 67 1D 2F 60 32 06 44 D3 34 87 80 CE 1B 8F 42 37 20 6E 3B 6E D0 20 73 48 D5 8B 22 3C	Difference between correct and incorrect ciphertests 05 B0 23 EE 37 AB 7F 31 D7 79 F4 DE 42 5B 90 F5 FA D0 B5 81 72 18 E3 7C F4 48 A1 3A 41 EE 5A 65
Correct modulus 51 68 A0 CC 86 A1 38 90 71 E8 83 44 C2 87 F0 67 D9 A5 10 40 0C B7 5D 3D 47 B3 4C FA EE F0 97 60 F6 36 25 F4 78 DD 39 AD 7C E0 64 CD 3F EC EE DB 0A B7 22 FF E6 35 AA 18 E0 23 B6 A8 E9 2B 72 7D	Recovered value of prime number p: 5F 6A 86 6C 9D 83 A6 B4 C7 43 2E 37 5C 25 92 43 20 AF BA AF EE 71 12 C0 F0 E0 7B FD 5F 7E 46 6B

Generic countermeasure against 2OFA

Algorithm 3 Second Order Resistant CRT-RSA with lock	
Inputs: CRT RSA private key and message C **Outputs**: RSA signature M or error detection	
Steps 1-8 are the same as in Algorithm 1	
9. If C' !== C then *Erase M*	If computed and expected verification values are not equal erase the signature
10. *RES := M*	Re-write the signature in a new variable RES
11. if C' == C then *Erase RES*	Execute verification once again and erase a new variable if verification fails
12. return RES	Return a value of a new variable

Are 3OFA feasible?

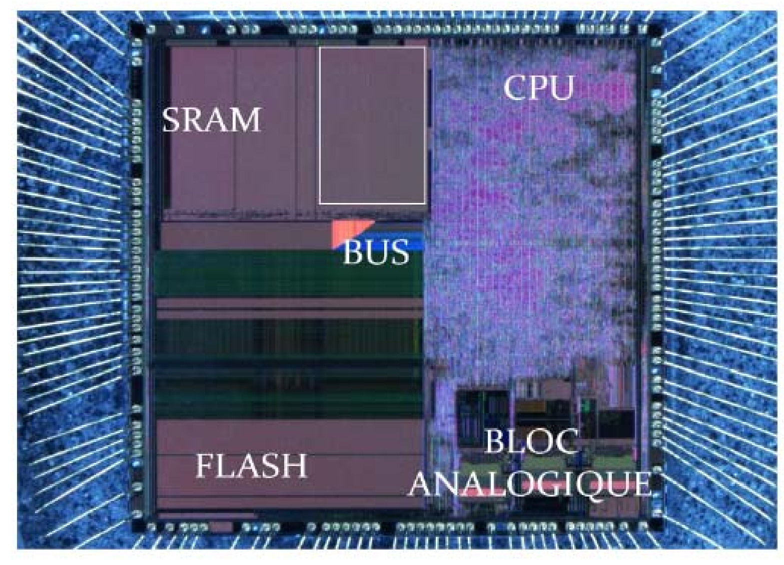
- E. Dottax, C. Giraud, M. Rivan, Y. Sierra, "On Second-Order Fault Analysis Resistance for CRT-RSA Implementations", WISTP'09
  - "Lock principle" – a check is performed twice in a rapid succession while inserting in between a simple but vital statement
- This countermeasure works so far
- Two shots fired in a rapid succession produce the same type of errors

Final words

This rectangle has approximately size 80x270 μm, while the spot which gives instruction skipping is about 80x40 μm. The size of the chip is 4000x4000 μm, so the vulnerable spot takes 0,135 % of the SoC. The laser aperture was about 35x35 μm. It was focused, so it is impossible to understand the actual size of the laser spot. Taking into the account all sizes it is possible to understand, why it was difficult to find this position.

Shooting through metal: UV

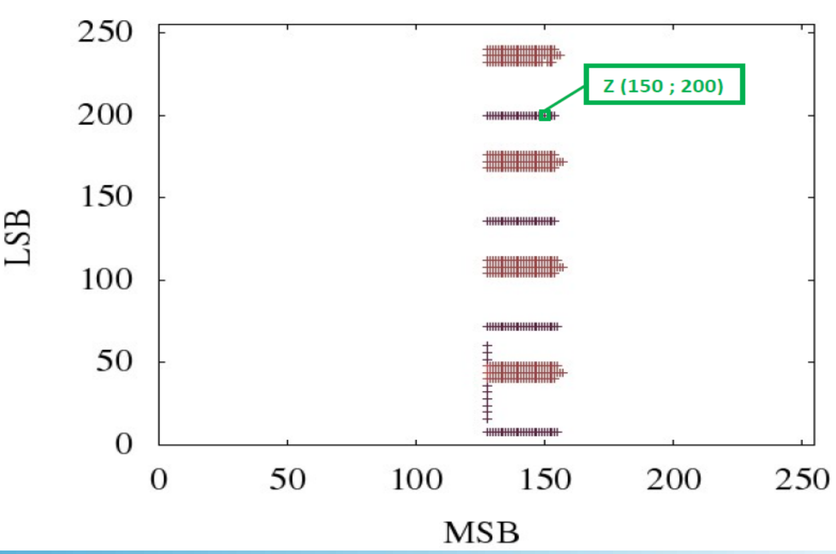
- SRAM was scanned with UV
  - Aperture 100%
  - Energy from 5% to 100% with step 0.2



0X00→0000 0000 et 0X80→1000 0000

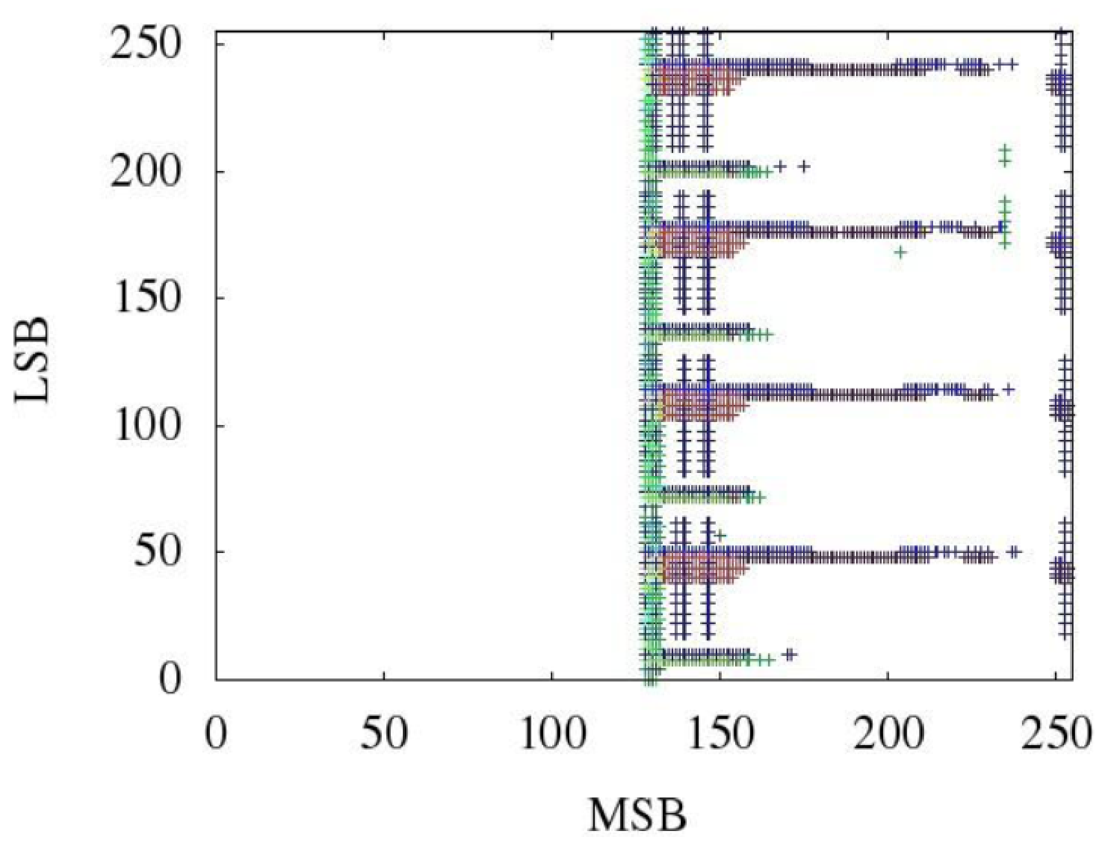
0XFF→ 1111 1111 et 0XBF→1011 1111

0XFF→ 1111 1111 et 0X7F→0111 1111

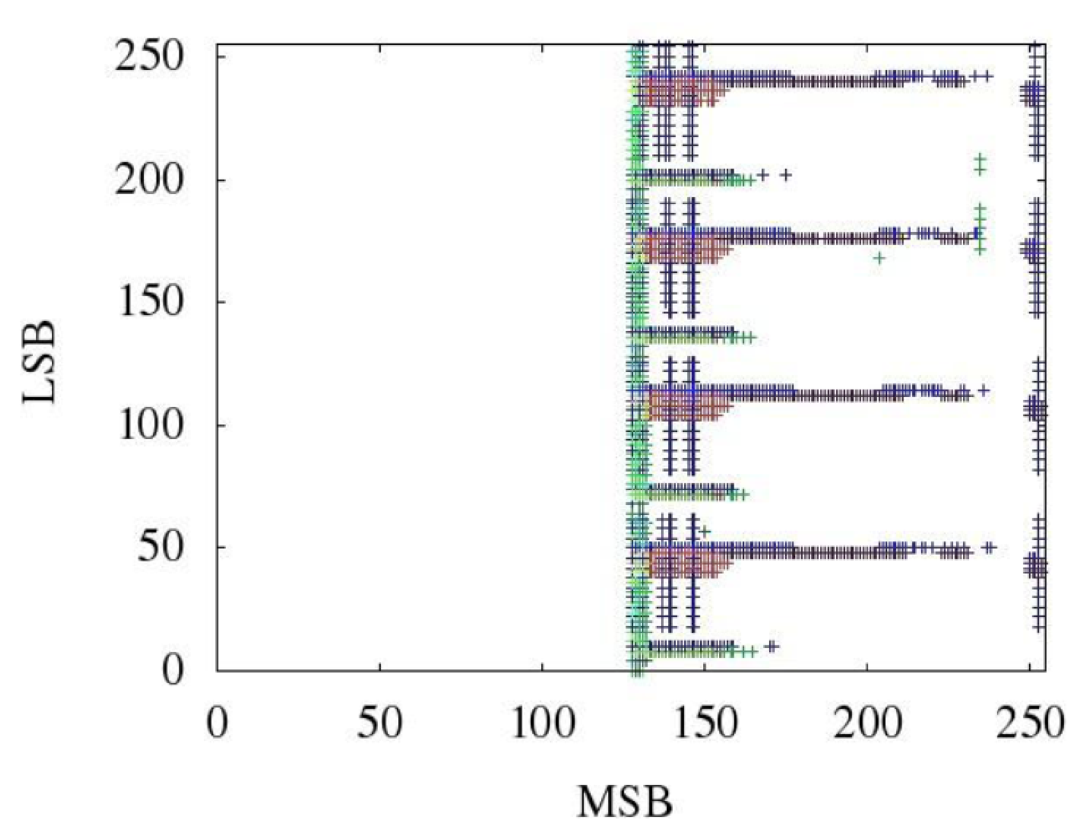


Shooting through metal: IR

- Scanning SRAM with IR laser
  - Aperture 20x20%
  - Energy 10%



Shooting through metal: green



- SRAM scan with a green wavelength
  - Aperture 20x20%
  - Energy 10%

Laser effect on IC

Drain (Gnd)

n^+ diffusion

$\vec{E}$

Courant en mA

Imax

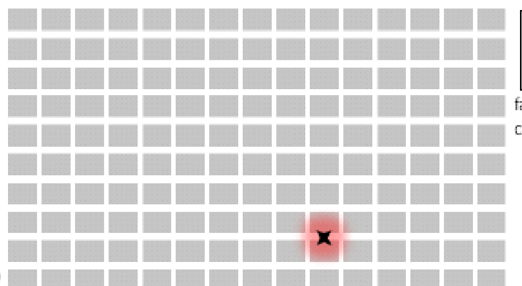
Composante instantanée

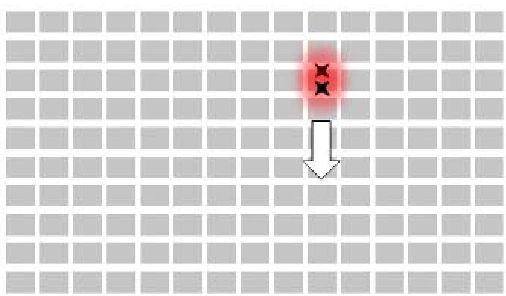
Composante retardée

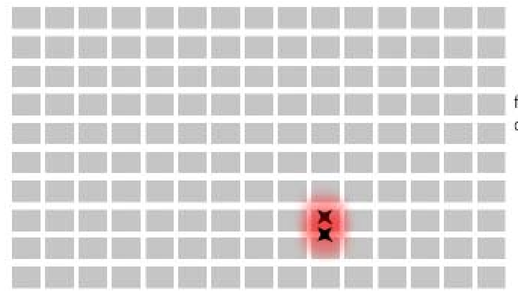
0

0,2 0,4 0,6 0,8 1

Temps en ns

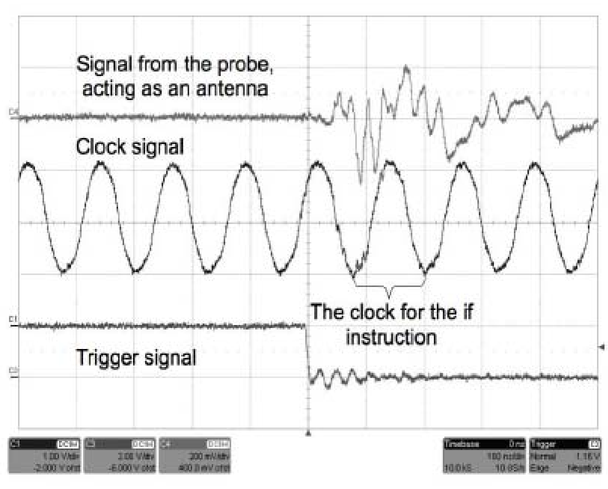


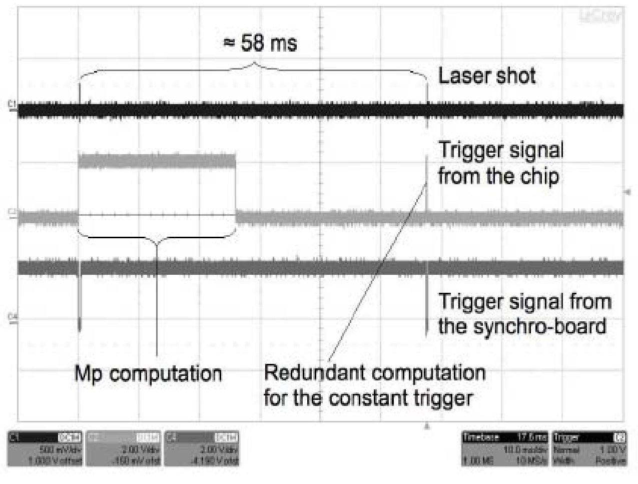




Parameter	Possible Values
location	no control, loose control, or complete control
timing	no control, loose control, or precise control
number of bits	single faulty bit, few faulty bits, or a random number of faulty bits
fault type	stuck-at fault, bit flip fault, random fault, or bit set or reset fault (to be defined in Section 1.3)
probability	(various possible values)
duration	destructive, permanent, or transient faults

First successful 2OFA with a laser





Example 2OFA on protected CRT-RSA

- ,,

Correct values	Faulty encryptions
e = 01 00 01 N = 51 68 A0 CC 86 A1 38 90 71 E8 83 44 C2 87 F0 67 D9 A5 10 40 0C B7 5D 3D 47 B3 4C FA EE F0 97 60 F6 36 25 F4 78 DD 39 AD 7C E0 64 CD 3F EC EE DB 0A B7 22 FF E6 35 AA 18 E0 23 B6 A8 E9 2B 72 7D M = 12 9D 1B F0 A5 29 A7 2E D6 06 6F 35 4D 7E 50 8F 51 3A 70 51 FA 7C 97 A9 63 74 04 03 24 97 9C E1 53 F6 53 35 AB DE 30 45 17 7B F2 EE DF FF 18 A5 12 3B 8B 42 6B 74 9A 4B 20 8D 0D 18 4E 7A F8 B0 C = 3A 7A 11 F7 04 FE 17 AD 5A BB 2D A1 18 6A 01 F6 41 41 54 66 1D 1E 17 47 10 E0 65 FA 31 A7 2E A9 0B FF 04 54 5E 15 3B 9F 28 89 07 E8 0B 84 17 81 76 12 7D 28 4B ED 5A 18 62 47 07 3C 1E 2E 3D 79	C'1 = 4B E2 61 70 9F B7 70 25 62 2A 01 5F 90 8C BD 8D DD 8D E4 1E 30 9D 68 AD 84 62 2C 33 8E 47 21 22 64 1D ED B6 42 1F 94 87 FA CF 6D 72 64 40 FF 56 21 0F C3 50 AB 91 0D 8A 95 DD 08 57 A0 B4 F2 1B C'2 = 11 5A AD CC 3E 9A B2 B8 93 1B 8E 61 B7 F9 7F C2 A6 FA B3 A8 2B 64 2C CE 63 80 84 F6 46 3F 6F FF EC 93 B8 73 1A 35 11 E7 8B 7E 78 80 B9 8D 75 97 AE EF DF 84 3C C1 84 39 D6 51 71 D2 9B 7D 18 B3 C'3 = 3F 79 3F 2C A5 AC 78 90 71 7C D6 64 A0 BB 66 66 AD 70 AA 87 7A E4 C6 16 75 4A AC 18 70 B1 6A 02 35 9D 52 61 E7 56 6D 3F 55 FA 7C CD 75 14 83 F7 78 85 DE A9 E6 A4 B9 54 E4 B6 C2 92 9E 25 34 76