UNSW SYDNEY

Hardware Security
Week 5 -
Fault Attacks

26T1
Hammond Pearce

Slide material acknowledgements to
Ramesh Karri, Benjamin Tan,
JV Rajendran, Jason Blocklove
Christian Pilato, Luca Collini

RECAP

1. Why are power analysis attacks so hard to prevent?

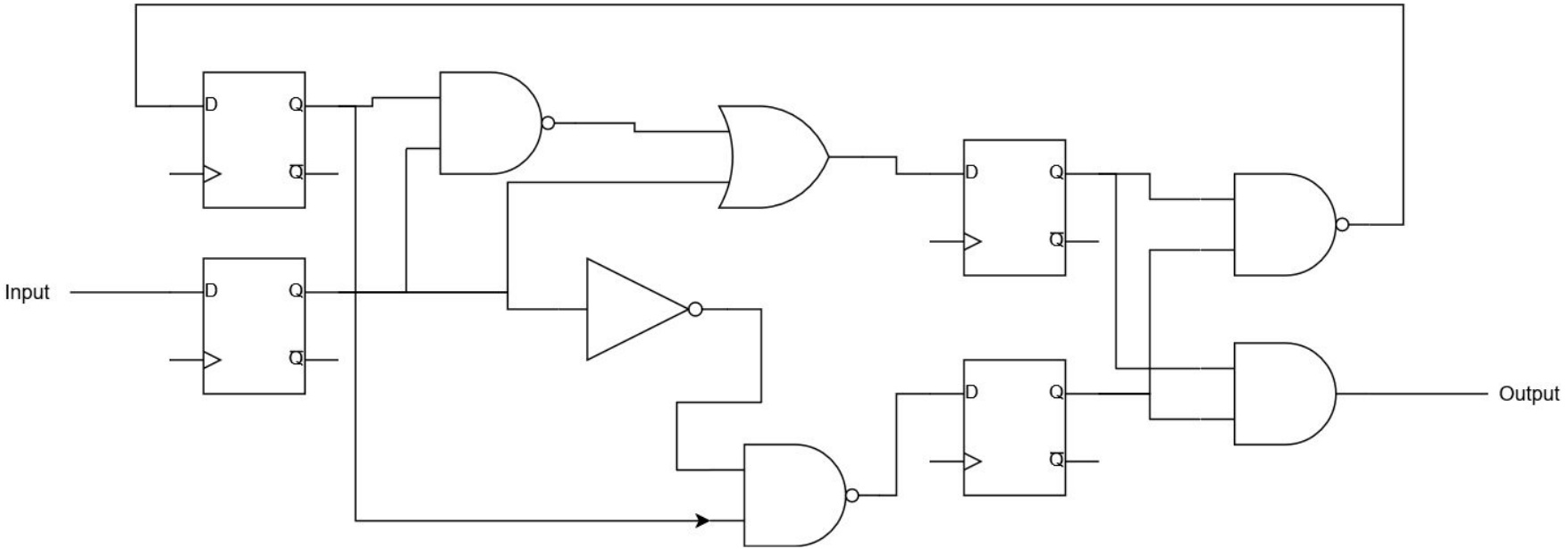
2. What are the differences between SPA, DPA, and CPA?

3. What are some techniques to minimise the signals needed for power attacks?

Part 2: Fault Attacks

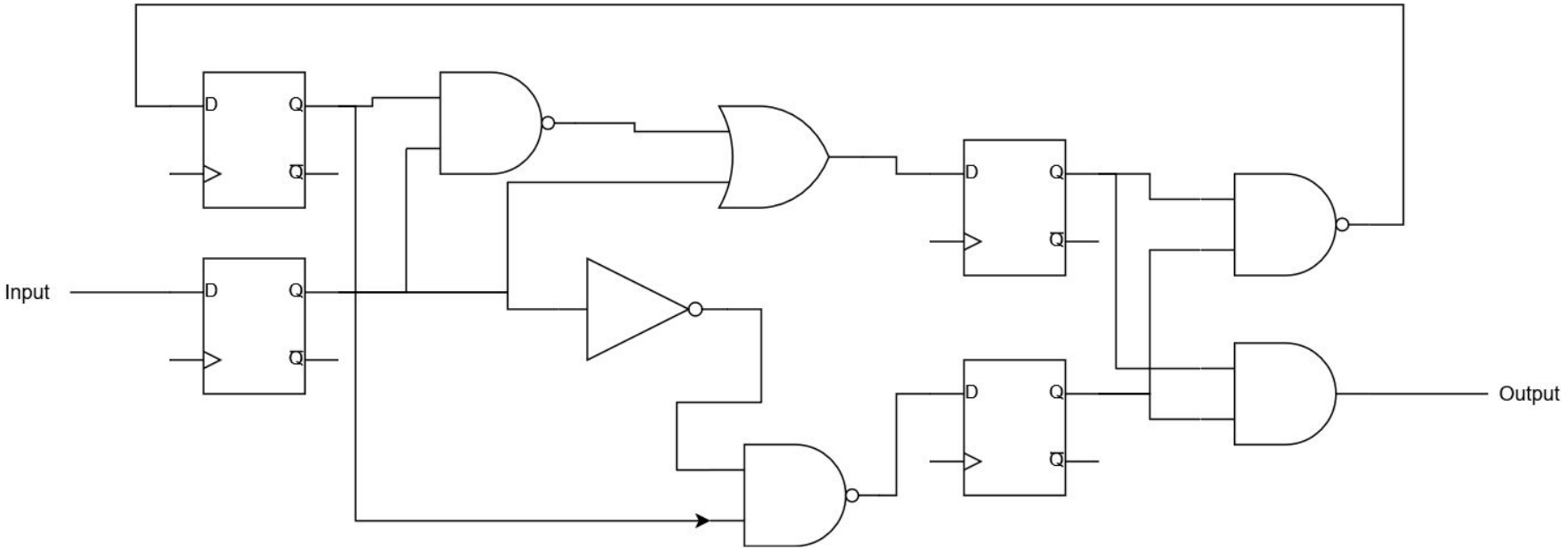
Let’s talk about hardware again

Circuits consume power, and take time to execute



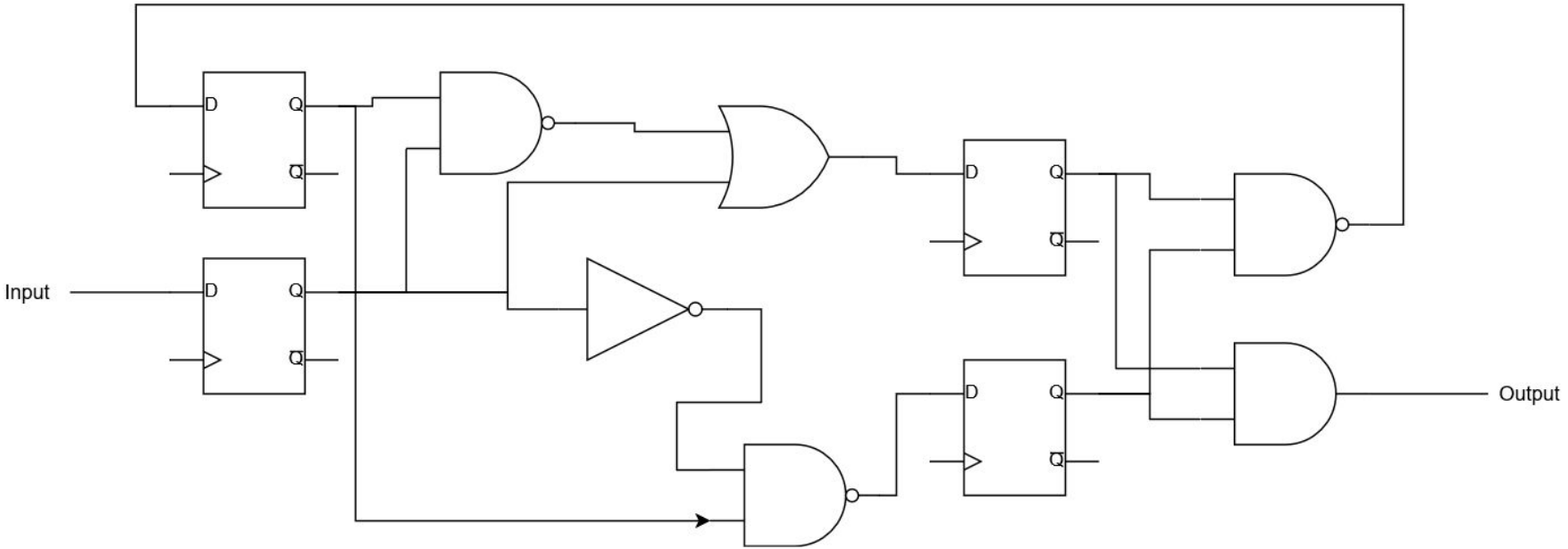
Let’s talk about hardware again

What if there is not enough power, or not enough time?

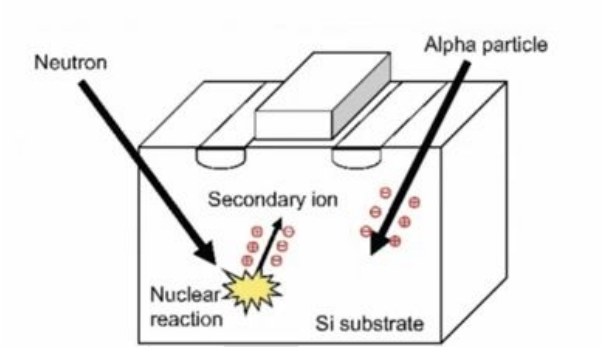


Let’s talk about hardware again

What if there is too much power? (How could this happen?)



Real-world faults



- Radiation (from spaaace!) can cause *random bit flips*

Real-world faults

ML model impacted by bit-flip

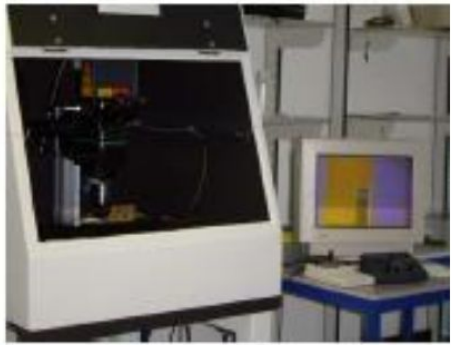
[DSN'21] Chen, Zitao et al. "A low cost fault corrector for deep neural networks through range restriction"

Faults can be added manually:

Deliberately:

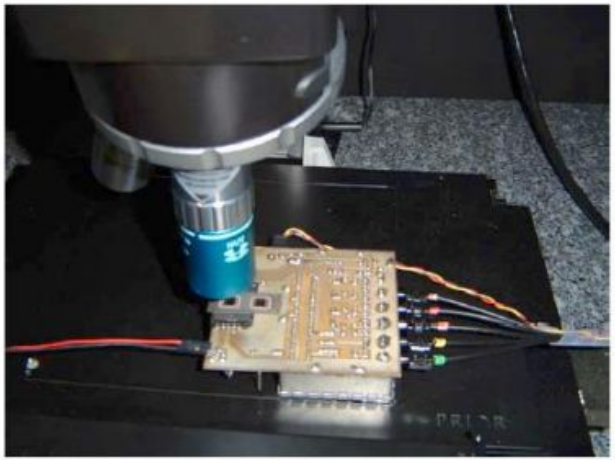
- Vary supply voltage
- Vary external clock
- Vary temperature
- Bright lights (White, laser)
- X-rays and ion beams
- Electromagnetic flux

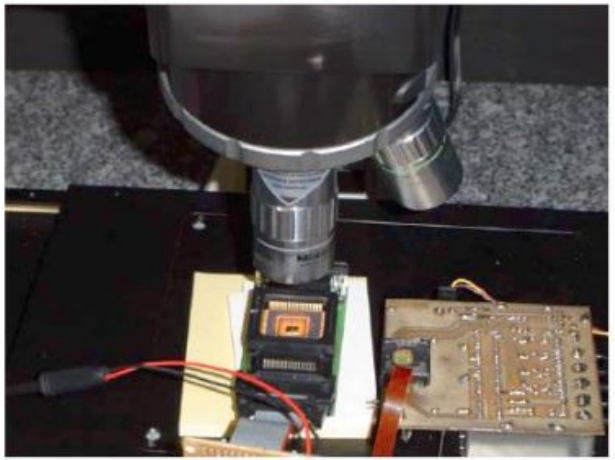
Can be expensive!



Can be expensive!

Laser



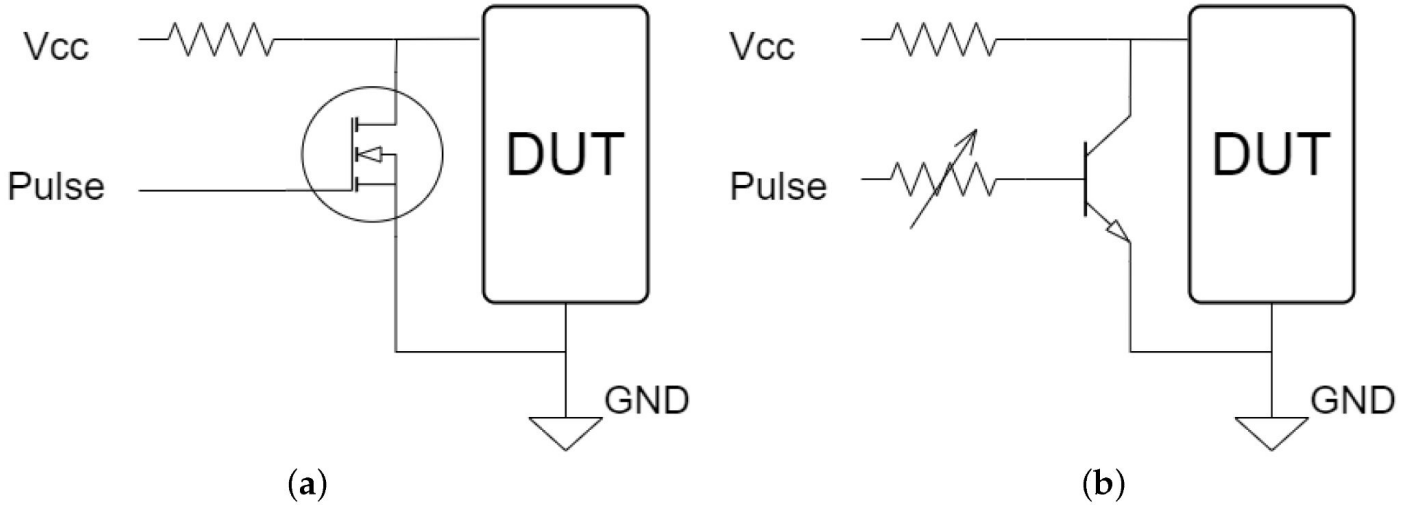


Fault injection via beams

- Need to control many parameters carefully
  - Spatial localization (x1, y1)
  - Firing window size (x2, y2)
  - Light/laser intensity
  - Wavelength

- Finding these parameters is not easy!

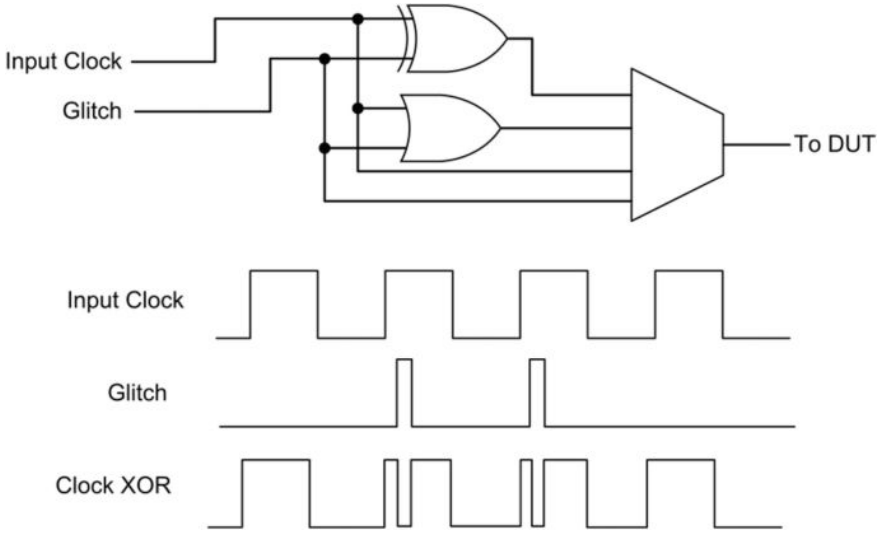
But fault injection can sometimes be easy



- How does this work?
- (Not all circuits vulnerable to this method)

Figure from Ruminot et al, “A Novel Approach of a Low-Cost Voltage Fault Injection Method for Resource-Constrained IoT Devices: Design and Analysis“, Sensors, 2023, https://www.mdpi.com/1424-8220/23/16/7180

…Fault injection can sometimes be easy

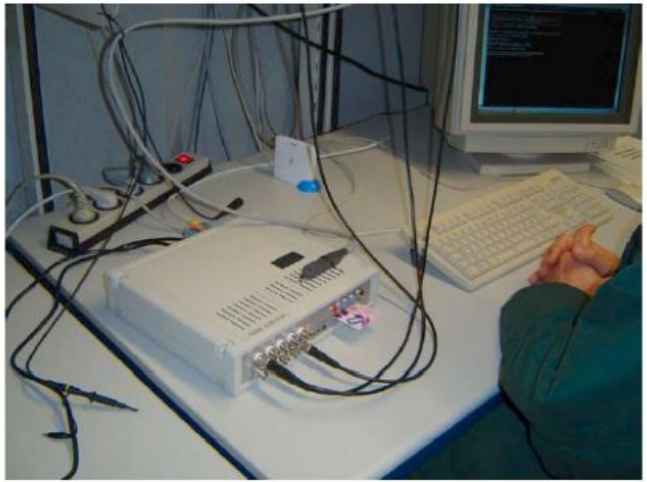


- How does this work?
- (Not all circuits vulnerable to this method)

Figure from Tehranipoor et al, “Clock Glitch Fault Attack on FSM in AES Controller”, https://link.springer.com/chapter/10.1007/978-3-031-31034-8_11

Fault injection setup

CLIO Glitch Injector

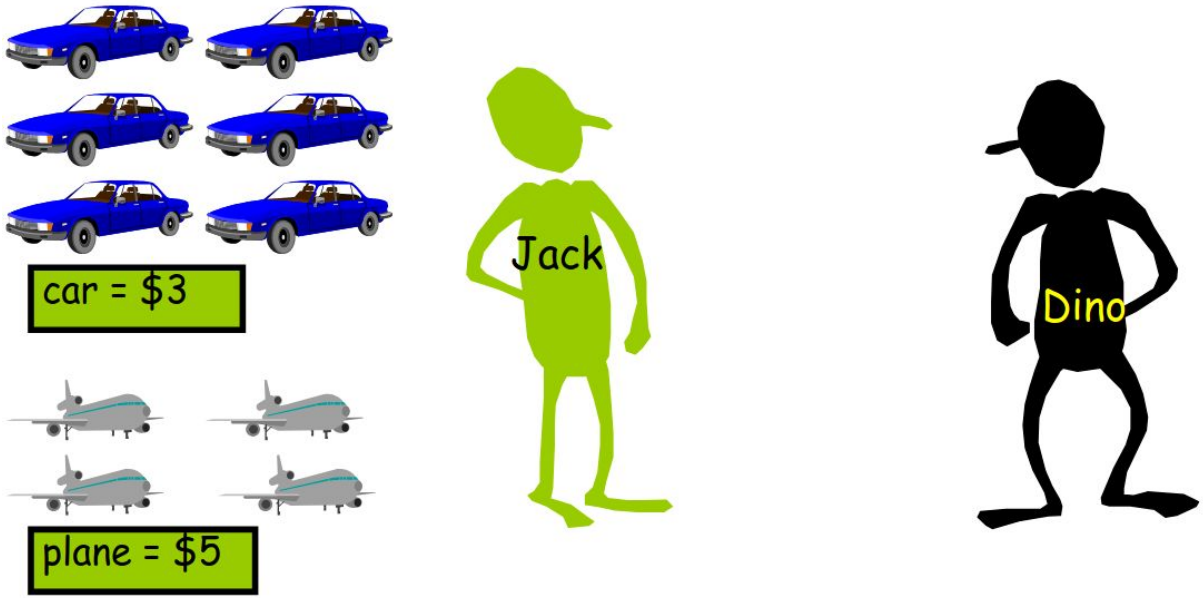


Why is this useful?

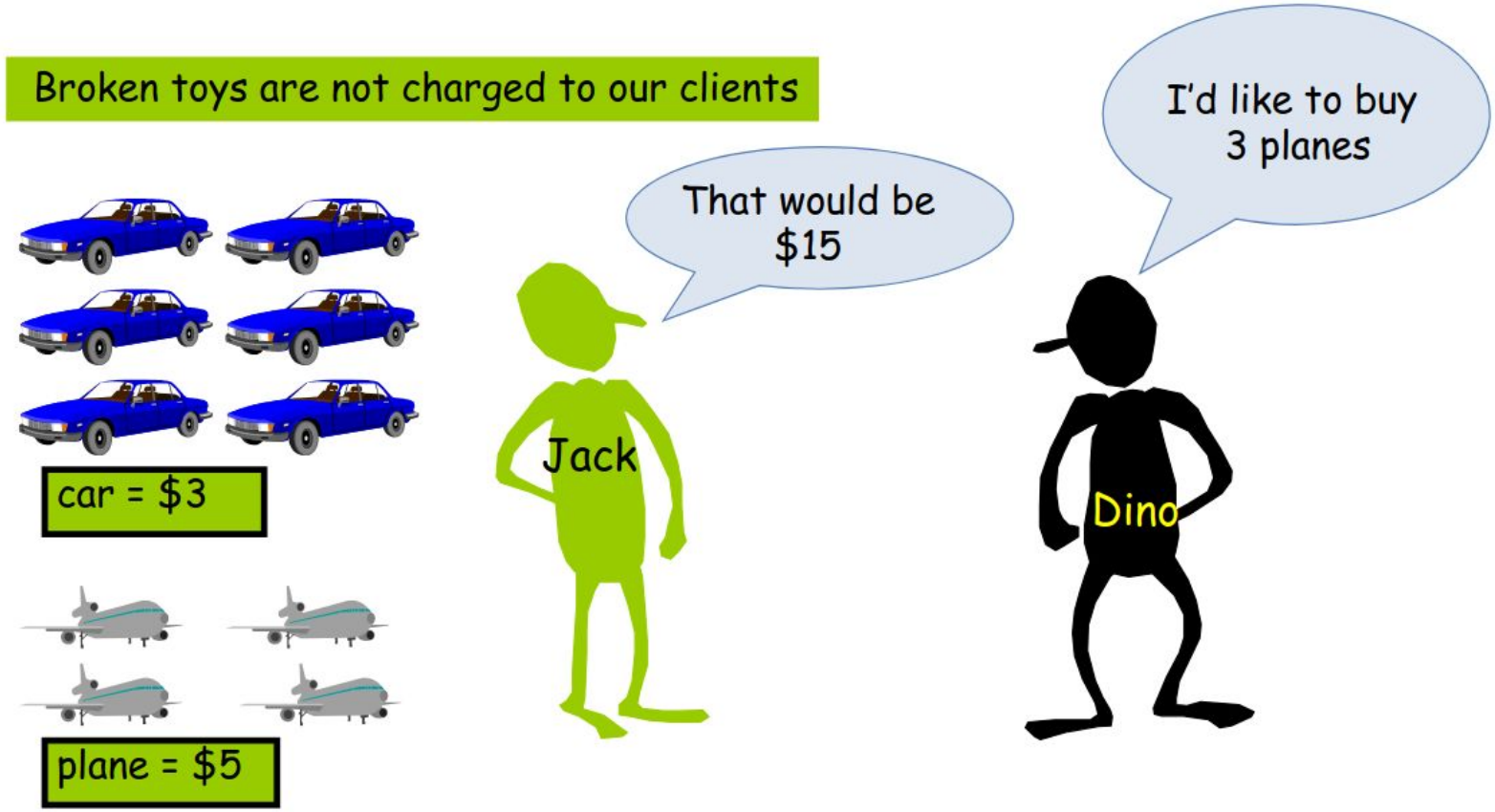
Fault injection scenario

Source: David Naccache

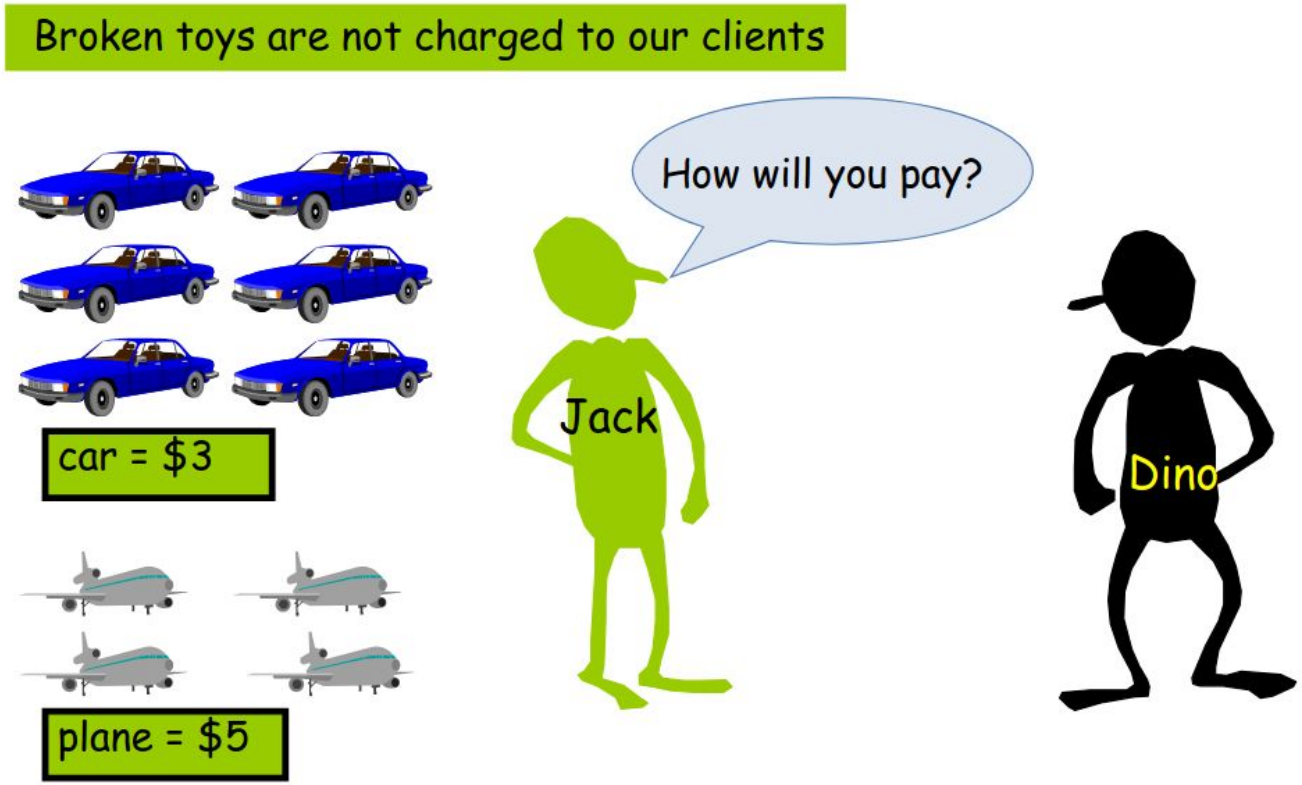
- Dino wants to buy toys from Jack
- Jack does not charge for toys broken in transit



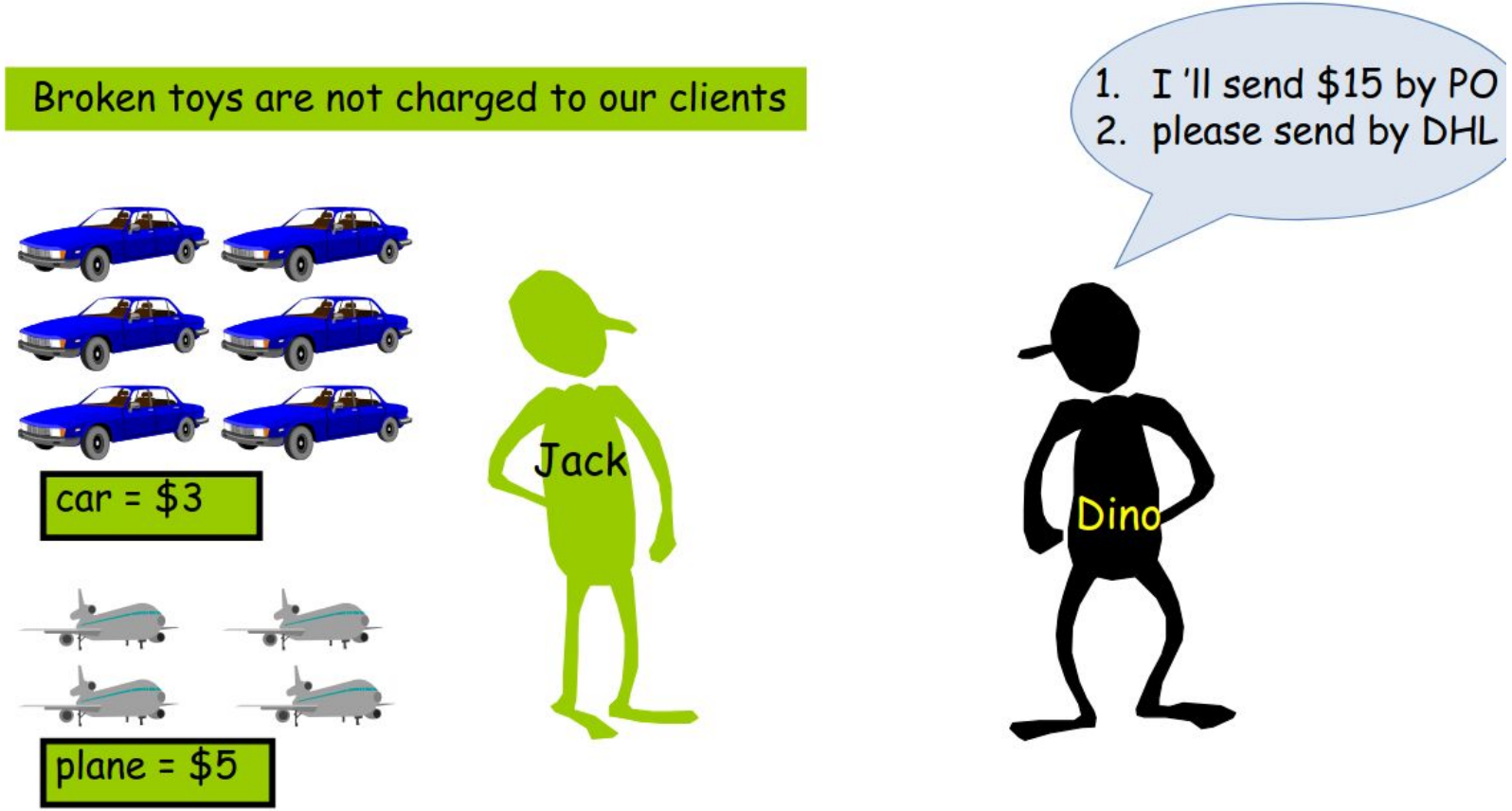
Fault injection



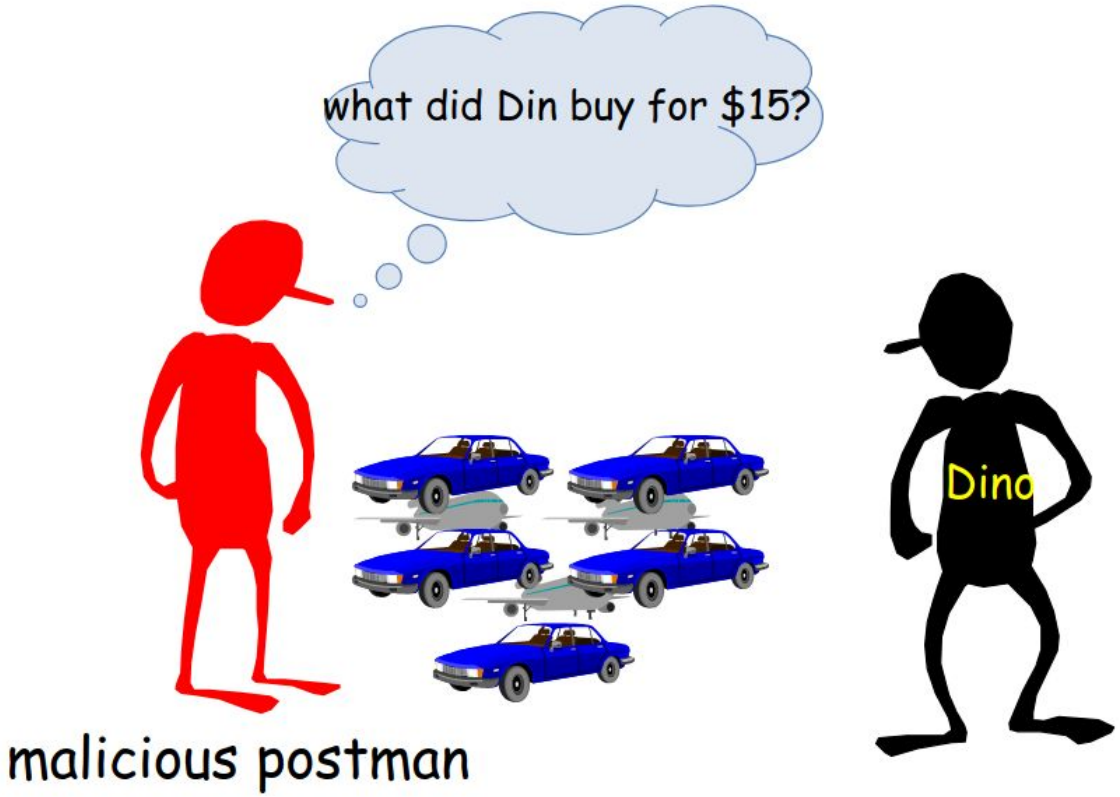
Fault injection



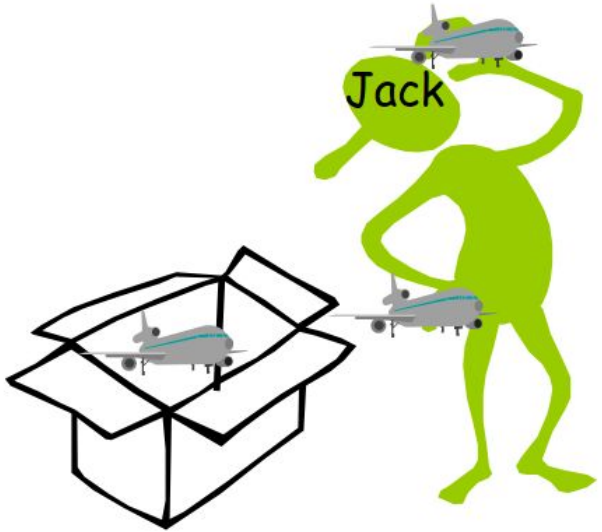
Fault injection



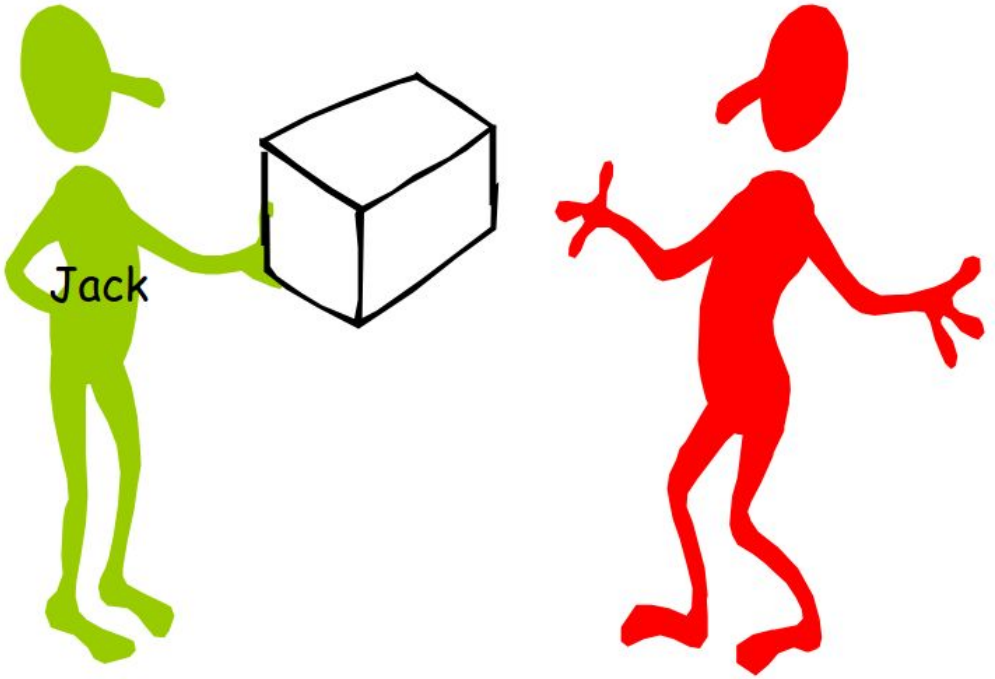
Postman: What did Dino buy for $15?



Jack prepares DHL package

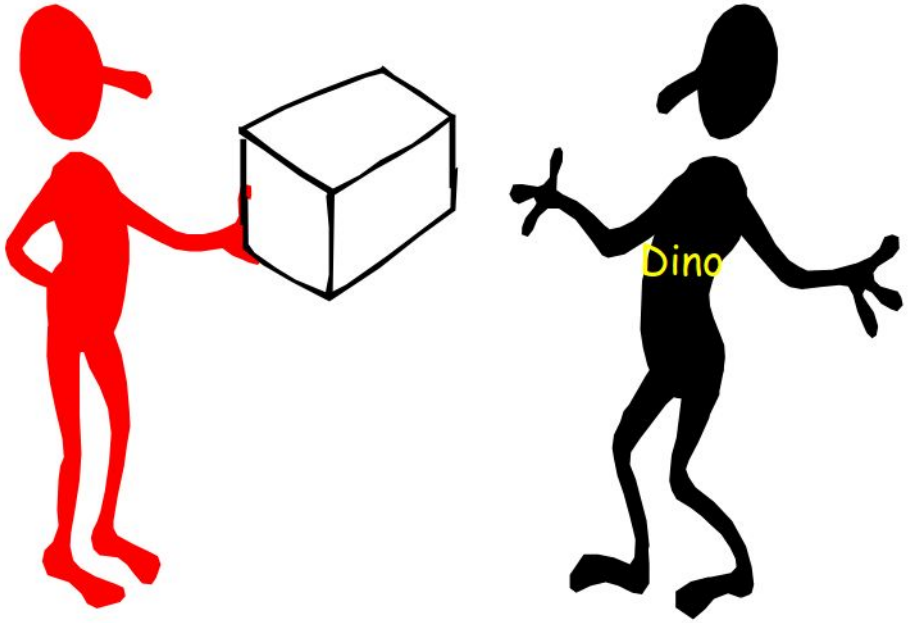


And gives it to the postman



Who kicks it … and breaks ONE toy

And delivers the package to Dino



A week later, Postman monitors Dino's PO

- It was $10 (don't pay for broken toy)
- Previous: $15, current: $10, one toy broken

A(cars * 3) + B(planes * 5) = $15
C(cars * 3) + D(planes * 5) = $10

A + B = C + D + 1

A, B, C, D - all integers >= 0

Solve:

A week later, Postman monitors Dino's PO

Only two possibilities add to $15 (five cars or three planes)

= 4 × 3 = $12

= 2 × 5 = $10

- The fault attack lets us know which option it was!

Fault attacks

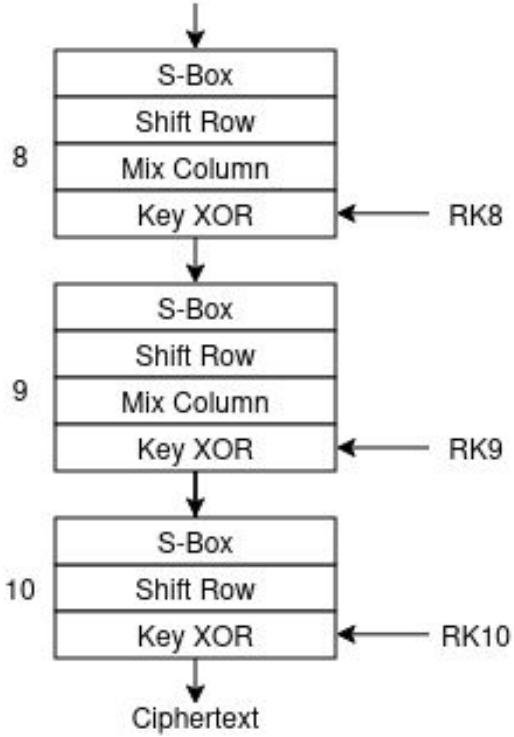
- Introduce an error in the device
- Perform encryption with and without presence of fault
- Use this to recover sensitive information!

Fault injection on AES

- Faults corrupt one byte of the AES intermediate state
- Specific byte known, but exact fault unknown (e.g. which bit(s))
- Need pairs of correct and faulty ciphertexts

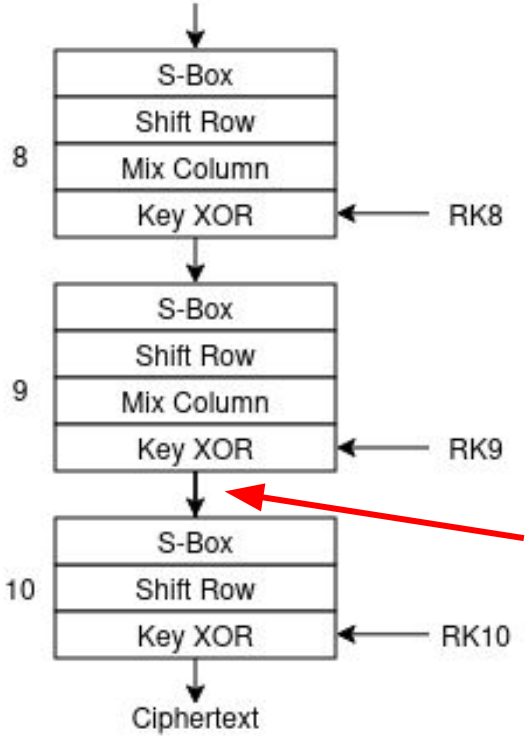
AES - where is the fault?

- A fault can cause a register bit flip
- We can target the AES text register
- We want to make our life easy



AES - where is the fault?

- A fault can cause a register bit flip
- We can target the AES text register
- We want to make our life easy
- Target end of encryption:
  minimise # of changes from fault



AES - round 10 fault

- Same PT is encrypted twice
- One encryption is correct, one is faulty
- Fault perturbs one byte of internal state
- Fault occurs during last round
- Attacker gets:
  - Correct ciphertext C
  - Faulty ciphertext C^*

then,

- $\Delta C = C \oplus C^*$

10th round intuition

- If fault happens before 10th round S-Box
  - Only one byte enters incorrectly
  - ShiftRows only permutes positions
  - KeyXOR only xors key bytes
  - No MixColumn: effect is one-byte local

- Only one ciphertext byte differs!
- That differing byte corresponds to one last-round key byte

Mathing it out

- For one affected byte position j

$$C[j] = S(x)[j] \oplus K10[j]$$

$$C^*[j] = S(x \oplus e)[j] \oplus K10[j]$$

- where :
  - C is ciphertext, C^* is faulty ciphertext
  - S is S-box
  - x is the input to round 10
  - e is the fault
  - $K10[j]$ is the 10th round key byte

What can we do with this?

- Choose a candidate byte k
- Run backwards:

$$InvS(C[j] \oplus k) = x$$

$$InvS(C^*[j] \oplus k) = x \oplus e$$

- For the correct key guess,

$$InvS(C[j] \oplus k) \oplus InvS(C^*[j] \oplus k) = e$$

But aren’t there too many unknowns?

- Yes!

But aren’t there too many unknowns?

- We can’t identify e from a single fault pair
- But if we can get one faulty pair, we can get another…
- We need a fault model:
  - Constant fault value
  - Bit-flip model
  - Stuck-at model
  - Small fault set
  - Some other repetition set…
- Assuming we have a fault model, we make some assumptions

Try identify the fault type:

- Keep reapplying the same plaintext
- Keep triggering the fault
- Does it only ever oscillate between two values $C[j]$ and $C^*[j]$?
  - This is a fixed-fault model

- Does $C^*[j]$ change over many different values?
  - The fault is inconsistent
  - This will be more difficult to attack (but not impossible)

Different fixed-fault types:

- Single bit-flip only
- Stuck-at-0 on one bit
- Stuck-at-1 on one bit

For each key guess

- compute the implied e
- reject keys who's implied e is impossible

Assuming “fixed-fault” model

- Fault stays the same when triggered
- Apply two plaintexts $PT1$, $PT2$, leading to Ciphertexts $C1[j]$, $C2[j]$
- Apply fault to get $C1^*[j]$, $C2^*[j]$, then:

$$d1(k) = InvS(C1[j] \oplus k) \oplus InvS(C1^*[j] \oplus k)$$

$$d2(k) = InvS(C2[j] \oplus k) \oplus InvS(C2^*[j] \oplus k)$$

Assuming “fixed-fault” model

- Fault stays the same when triggered
- Apply two plaintexts $PT1$, $PT2$, leading to Ciphertexts $C1[j]$, $C2[j]$
- Apply fault to get $C1^*[j]$, $C2^*[j]$, then:

$$d1(k) = InvS(C1[j] \oplus k) \oplus InvS(C1^*[j] \oplus k)$$

$$d2(k) = InvS(C2[j] \oplus k) \oplus InvS(C2^*[j] \oplus k)$$

- Given that $d1(k) == d2(k)$ due to the fixed fault…

Guess key values k from 0..255, and:

Apply $C1$, $C1^*$, $C2$, $C2^*$; match when $d1(k) == d2(k) \rightarrow k$ is a candidate

Apply plaintexts as needed $d3$, $d4$, $d5$… until only one possible answer!

What about not a fixed fault?

- As long as we can constrain the "e" in some way,
- We can construct suitable equations.
  - E.g, single bit-flip means $e \in \{0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80\}$
- We'll just have a different relationship between $d1(k)$, $d2(k)$, etc.

5 minute break (break 1)

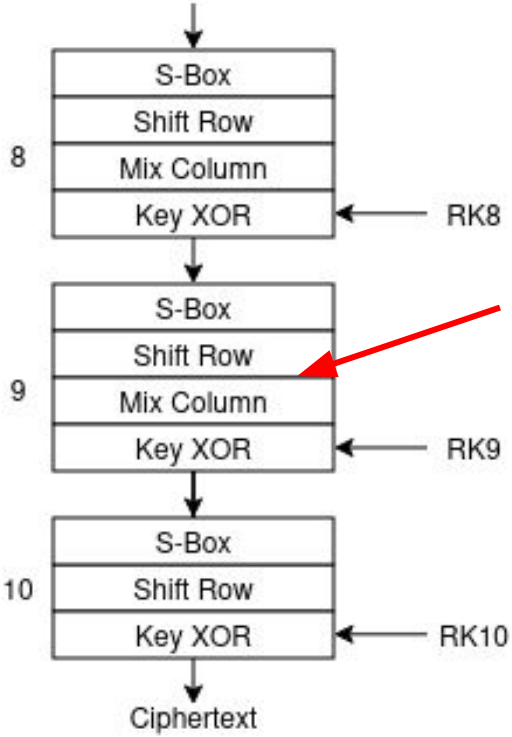
Talk to your neighbour

Are you ready for flex week

We're all ready for flex week

9th round MixColumns

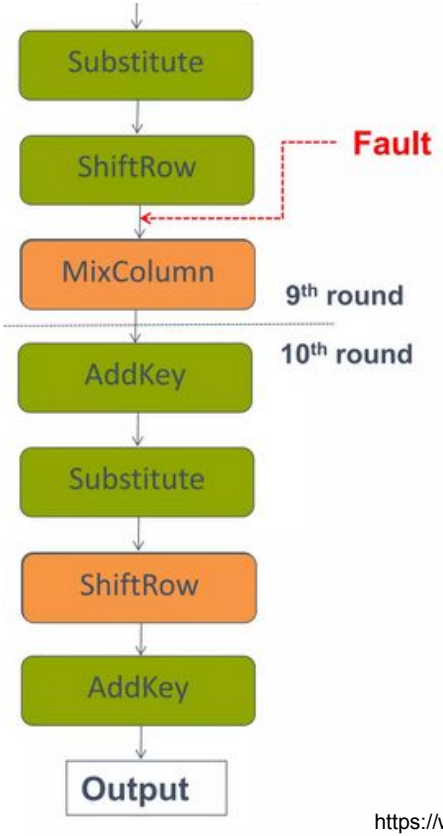
- Slightly more complex!
- Two keys involved (RK9, RK10)
- MixColumns spreads the fault to 4 bytes



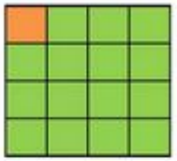
These differences are not arbitrary!

They satisfy the AES column diffusion relation.

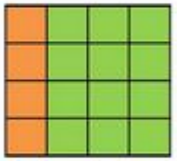
Fault impact on AES



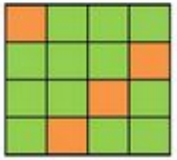
- Inject fault before MixColumn
- Fault changes chosen byte



- Mixcolumn propagates fault



- ShiftRow propagates fault to 4 cells
- One fault affects 4 output bytes

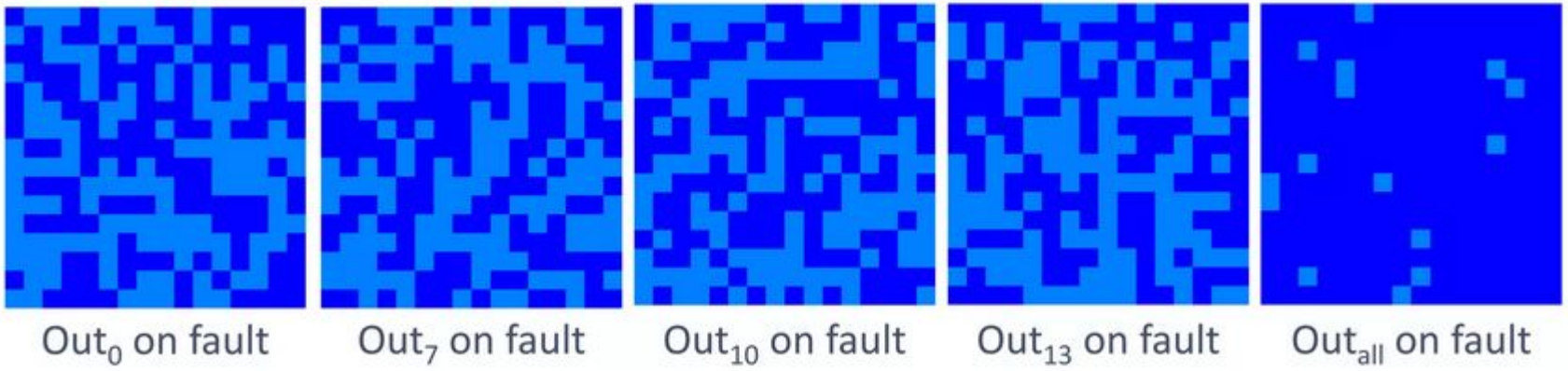


https://www.slideshare.net/slideshow/practical-differential-fault-attack-on-aes/79686890

Finding the fault value

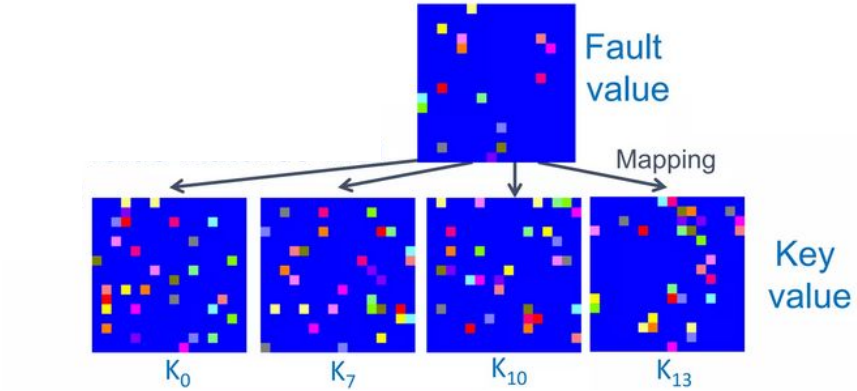
Fault in specific byte propagates to 4 output bytes

Each fault pair (correct, faulty output) halves number of possible values for random fault



A group of 4 affected output bytes reduces possible faults to ~15!

Finding the key value



- A specific fault value matches with two key values
- Four faults can break the key
  - 4 x 32 reduced to 4 x 8 bits
  - Remaining entropy is 32 bits
  - Can be brute forced

Why does this work?

- MixColumns takes a 1-byte error “e”,
- and expands it to a column:

(2e, e, e, 3e)

- (it could also be one of its rotations,

(3e,2e,e,e)

(e,3e,2e,e)

(e,e,3e,2e)

- depending on which column faulted)

Rearranging Round 9 equations

Given byte positions j0, j1, j2, j3

For a guessed last-round subkey bytes (k0, k1, k2, k3), define:

$$u0 = \text{InvS}(C[j0] \oplus k0) \oplus \text{InvS}(C^*[j0] \oplus k0)$$

$$u1 = \text{InvS}(C[j1] \oplus k1) \oplus \text{InvS}(C^*[j1] \oplus k1)$$

$$u2 = \text{InvS}(C[j2] \oplus k2) \oplus \text{InvS}(C^*[j2] \oplus k2)$$

$$u3 = \text{InvS}(C[j3] \oplus k3) \oplus \text{InvS}(C^*[j3] \oplus k3)$$

Rearranging Round 9 equations

- All four u values are not independent
- They must all come from the same e
- So if you know e, each byte could be solved

So…

Step by step for 9th round

1. Collect correct ciphertext C and some faulty ciphertexts C^*
   - if same PT works for different C^*, use it, if not (fixed fault), use another PT
2. For each (C, C^*) check:
   - Does difference have 4 non-zero bytes in positions for a single-column fault?
3. If so, map the positions to a round-9 column
4. Guess a value e in 1..255
   - Assign the expected difference value to each of the four positions
5. For each byte position ji, compute:

$$Ki(e) = \{ k \text{ in } 0..255 : InvS(C[ji] \oplus k) \oplus InvS(C^*[ji] \oplus k) = expected\_i(e) \}$$

Step by step for 9th round

6. Form candidate 4-byte tuples only from the Cartesian product:

$$K0(e) \times K1(e) \times K2(e) \times K3(e)$$

- This is usually tiny compared with 2^32

7. Repeat with a second fault pair and intersect candidate tuples.

- That usually collapses the ambiguity very quickly.

How many faults do we need?

- Round 9 faults spread in a very specific pattern
- This pattern is quite restrictive
- As such… Round 9 faults can be solved with very few pairs
- Often just 2!
- We also have fewer restrictions on the nature of the fault.

Why 9th round needs less data than 10th?

For round 10 with a static fault:

- the test is :
  - “does the inferred byte fault stay constant?”

For round 9:

- the test is stronger:
  - “does the recovered 4-byte vector match the AES MixColumns template,
  - and does it stay consistent across traces?”

- That strength is why round 9 often needs only two good pairs.

Can we do it with one faulty pair?

Can we do it with one faulty pair?

Inject fault in round 8

Fault randomly changes chosen byte

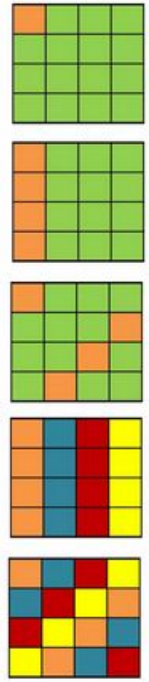
MixColumn propagates fault in column

ShiftRow propagates fault to 4 cells

MixColumn propagates 4 faults in 4 columns

ShiftRow propagates 4 faults to 16 cells, Exposing 16 byte bytes

One correct + fault pair + 32 bit brute force can reveal a 128-bit key!



Challenges with single-fault DFA

- Fault model must be known
  - Unknown byte hit?
    - Blind hit multiplies search space by 4
  - Unknown round hit?
    - Blind round hit multiplies search space by 10!
  - Unknown operation hit?
    - Out-of-model faults completely “mess up” the key search

- In practice:
  - A 32-bit AES brute force can be <20 minutes
  - With unknowns this can grow to days, or even be impossible

The more faulty outputs, the better

- If you can get one fault, you can probably get more
- Try select faults that match the fault model
- Then go to brute force

… all that math is hard

Fortunately, it is extremely well-studied.

There's even a Python library for it!
https://pypi.org/project/phoenixAES/

This library can solve Round 9 and Round 8 faults for you (but not round 10)

Can you guess what is coming next…?

Assessments: Week 5 (Fault Attack) - Lab 3

Lab 3

- Receive three unique AES FPGA bitstreams
- Round 8 fault, Round 9 fault, Round 10 fault
  - Use PhoenixAES for Round 8/9, your own mad skills for Round 10
- For each bitstream, interact over SPI
- When a button is pressed, encryption will be faulty
- This will allow you to collect (PT, C, C^*) triples.
- Use these to solve for the embedded keys in each core!

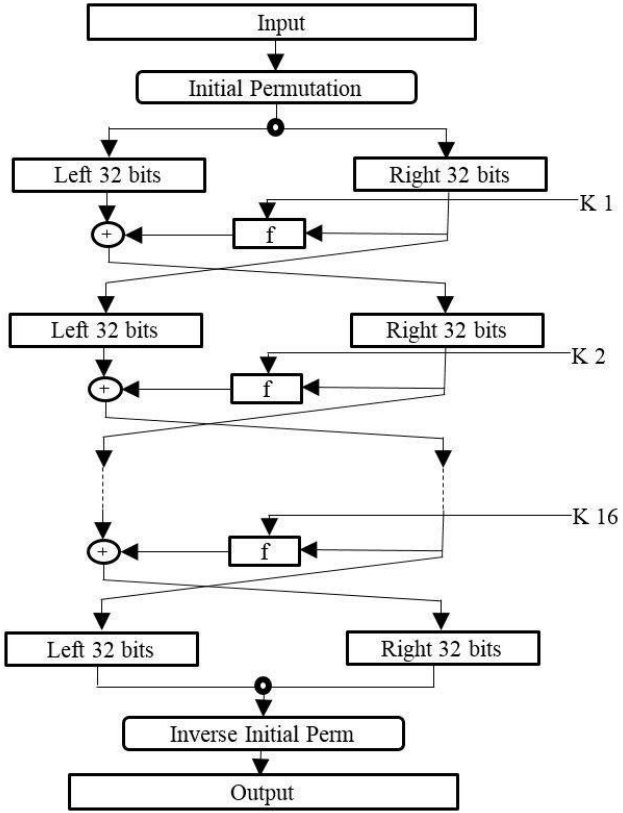
Released by Friday, Due Week 7

Appendix

Fault attack on DES

$L_{16} = R_{15}$

$R_{16} = P(S(E(R_{15}) \oplus K_{16})) \oplus L_{15}$

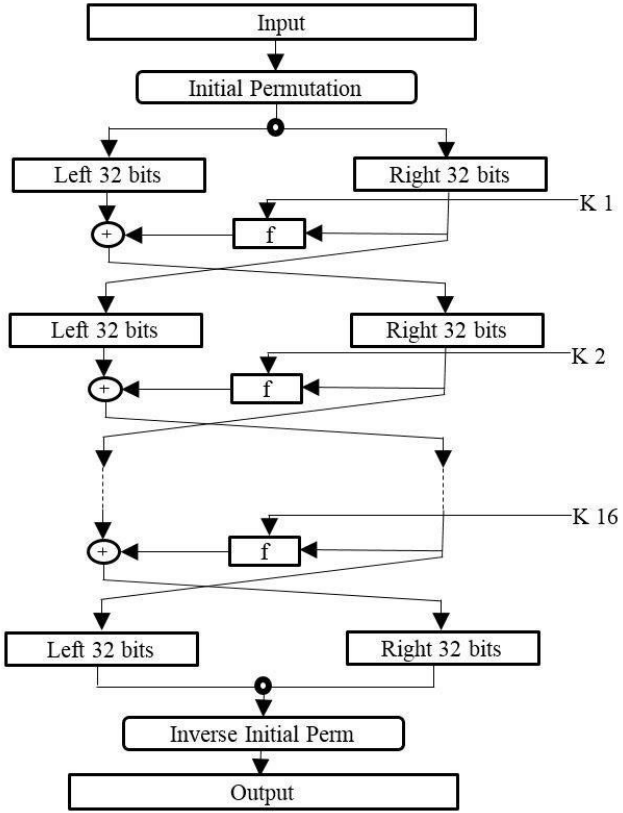


Fault attack on DES

$L_{16} = R_{15}$
$R_{16} = P(S(E(R_{15}) \oplus K_{16})) \oplus L_{15}$

- *Introduce a fault in R_{15}*

$R_{15}' = R_{15} \oplus e$
$L_{16}' = R_{15}'$
$R_{16}' = P(S(E(R_{15}') \oplus K_{16})) \oplus L_{15}$



Fault attack on DES

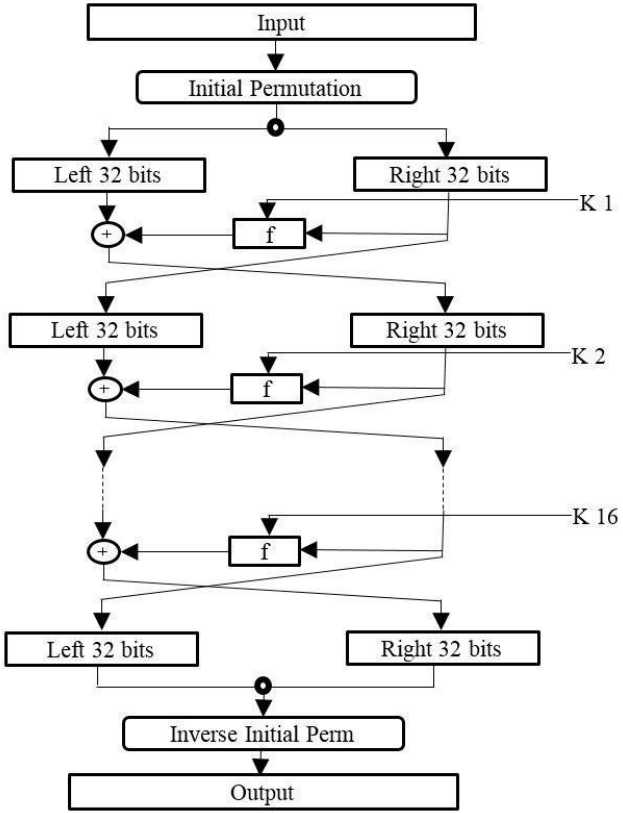
$L_{16} = R_{15}$
$R_{16} = P(S(E(R_{15}) \oplus K_{16})) \oplus L_{15}$

- *Introduce a fault in R_{15}*

$R_{15}' = R_{15} \oplus e$
$L_{16}' = R_{15}'$
$R_{16}' = P(S(E(R_{15}') \oplus K_{16})) \oplus L_{15}$

- *Rearrange*

$R_{16} \oplus R_{16}' = P(S(E(R_{15}) \oplus K_{16})) \oplus P(S(E(R_{15} \oplus e) \oplus K_{16}))$



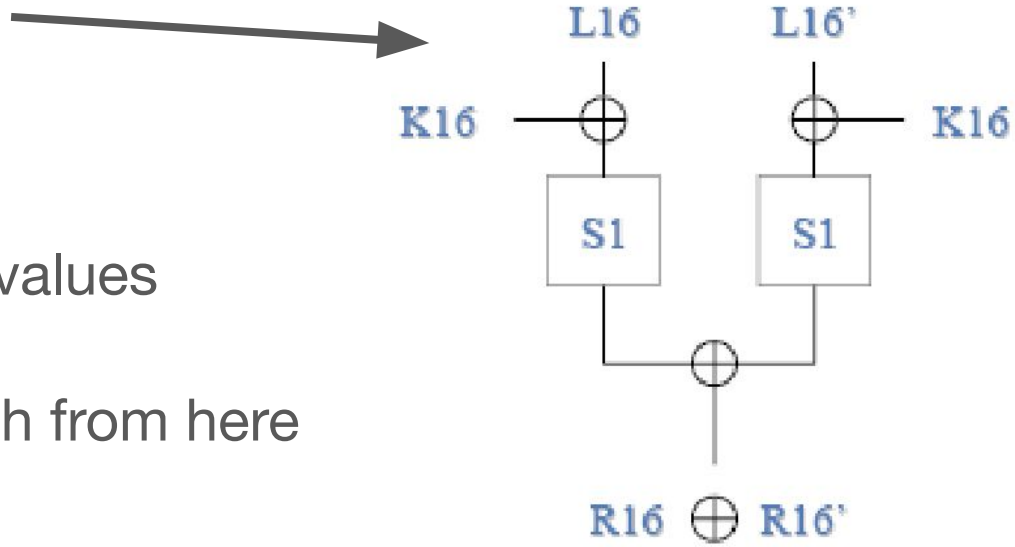
Fault attack on DES

$$R_{16} \oplus R_{16}' = P(S(E(R_{15}) \oplus K_{16})) \oplus P(S(E(R_{15} \oplus e) \oplus K_{16}))$$

- We know R_{16}, R_{16}'
- Brute force for K_{16} which satisfy the above relation!

15-th round DFA on DES

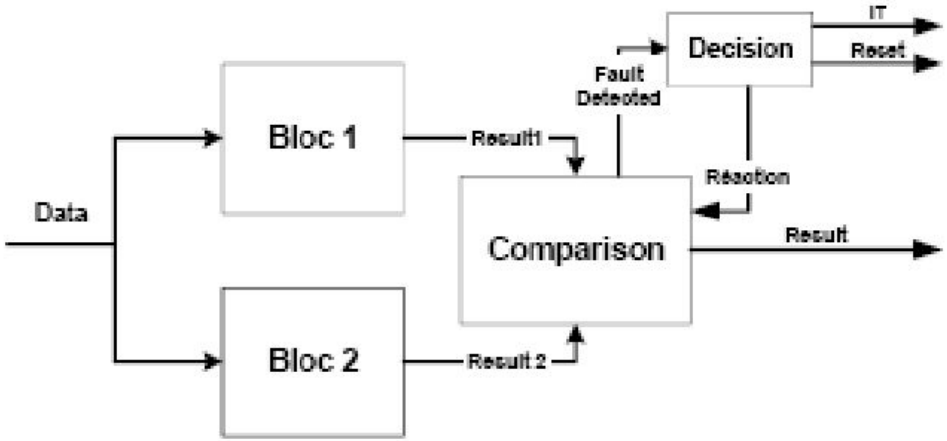
- For each S-Box, verify:



- Gives 2^{26} possible key values
  - Much better than 2^{56}!
- Do an exhaustive search from here

Defenses against fault attacks

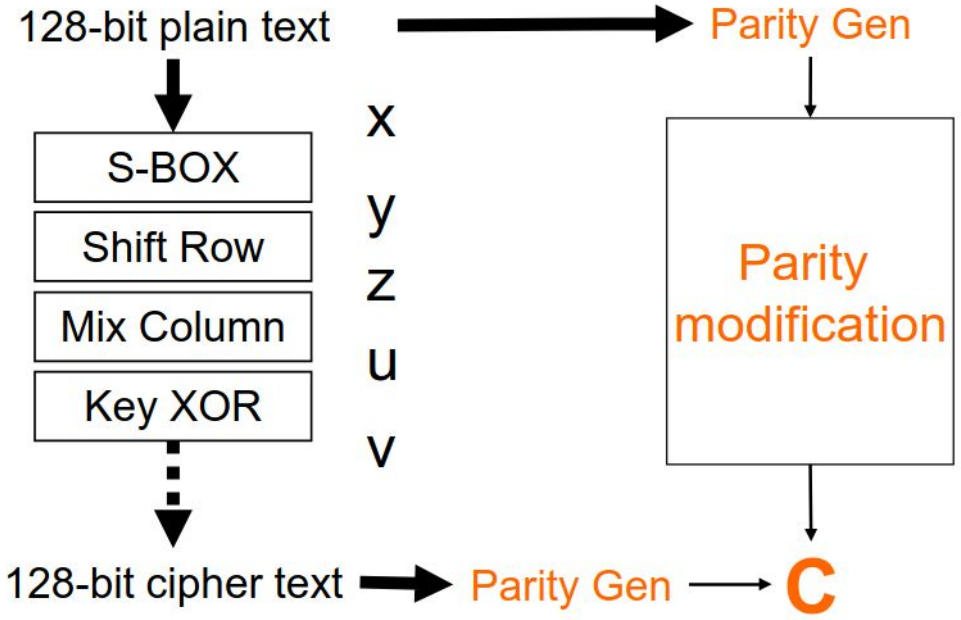
- Redundant hardware!!



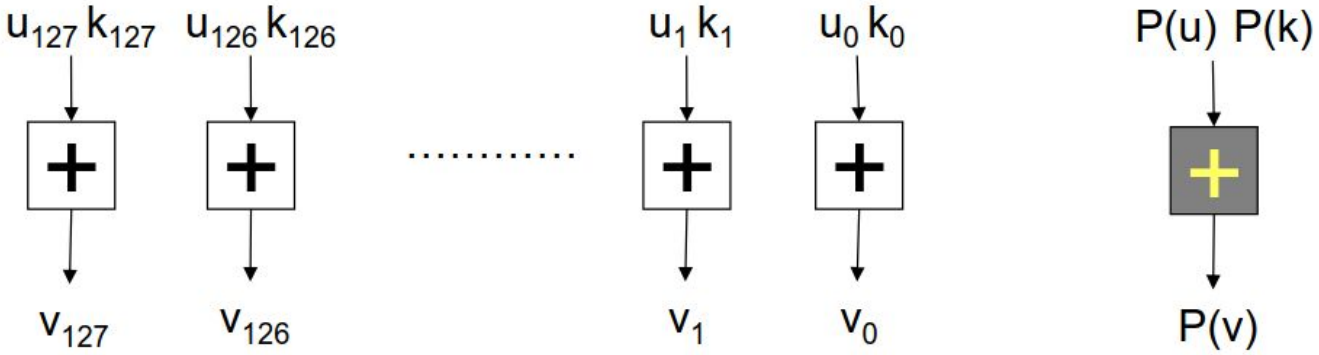
- Implement hardware twice and look for differences…
- Downsides: area overhead :(

Concurrent Error Detection (CED)

- Parity checks



Round operation: Key XOR



- 128-bit XOR does not preserve parity
- Output parity can be predicted as $P(v) = P(u) \oplus P(k)$
- P(k) can be pre-computed and stored with round keys
- No performance overhead

Round operation: Mix Column

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

$$X = \begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{bmatrix}$$

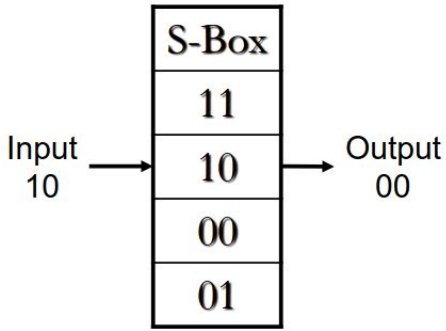
- Mix column operates over GF(2^8) matrix multiplication
- This operation preserves parity
- P(u) = P(z)
- No additional circuits needed

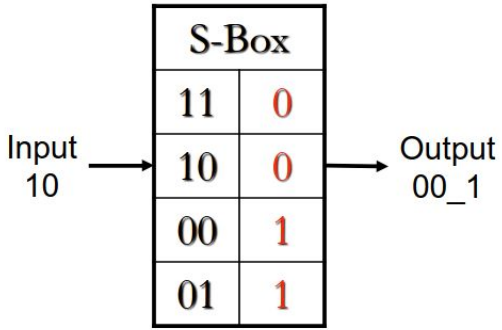
Shift row

- Each row rotates by a fixed number of bytes
- Rotations do not affect parity

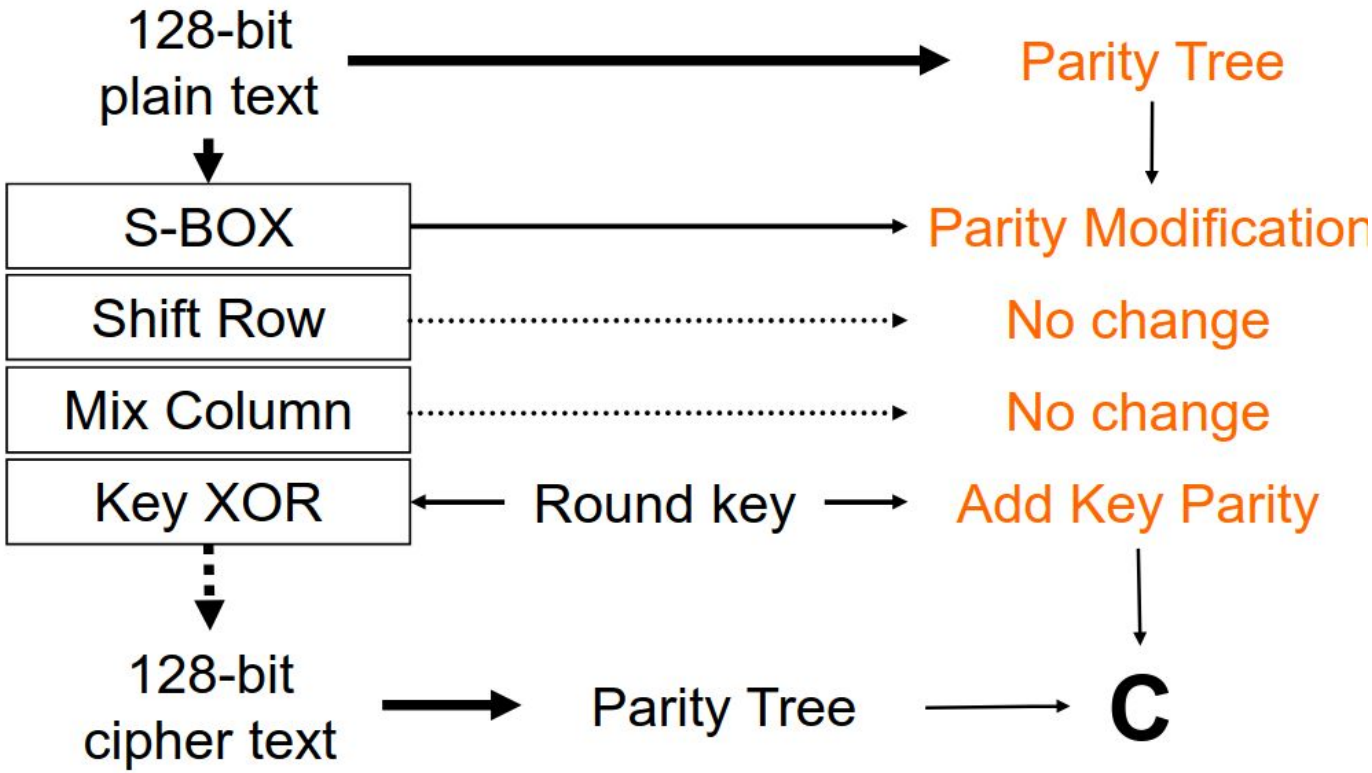
S-Box

- No parity relationship between input and output
- Need to add a parity LUT: one bit to each S-Box entry
  - 1 when parity of input (address) is different from output (content)
  - 0 otherwise
  - Pre-calculate when S-Box is designed
- No performance overhead, small area overhead





AES Parity based CED



AES Parity-based CED

- Low cost low latency parity technique
  - Extra bit added to S-Box locations
  - Shift Row preserves parity
  - Mix Col preserves parity
  - Key XOR adds parity of round key to parity of input

- FPGA implementation:
  - Performance and area overhead <10%

Takeaways

- Fault injection is a powerful technique to disrupt HW
  - Cause random bit flips
- In crypto HW allows you to produce simultaneous equations:
  - Produce pairs of (ciphertext, faulty ciphertext)
  - Use these to reduce brute force key search space
- Defenses exist:
  - Multiple copies of hardware (used in high radiation environments)
  - Error correcting codes/Runtime parity (Concurrent Error Detection)

3 minute break (break 2)

Talk to your neighbour

Are you ready for flex week

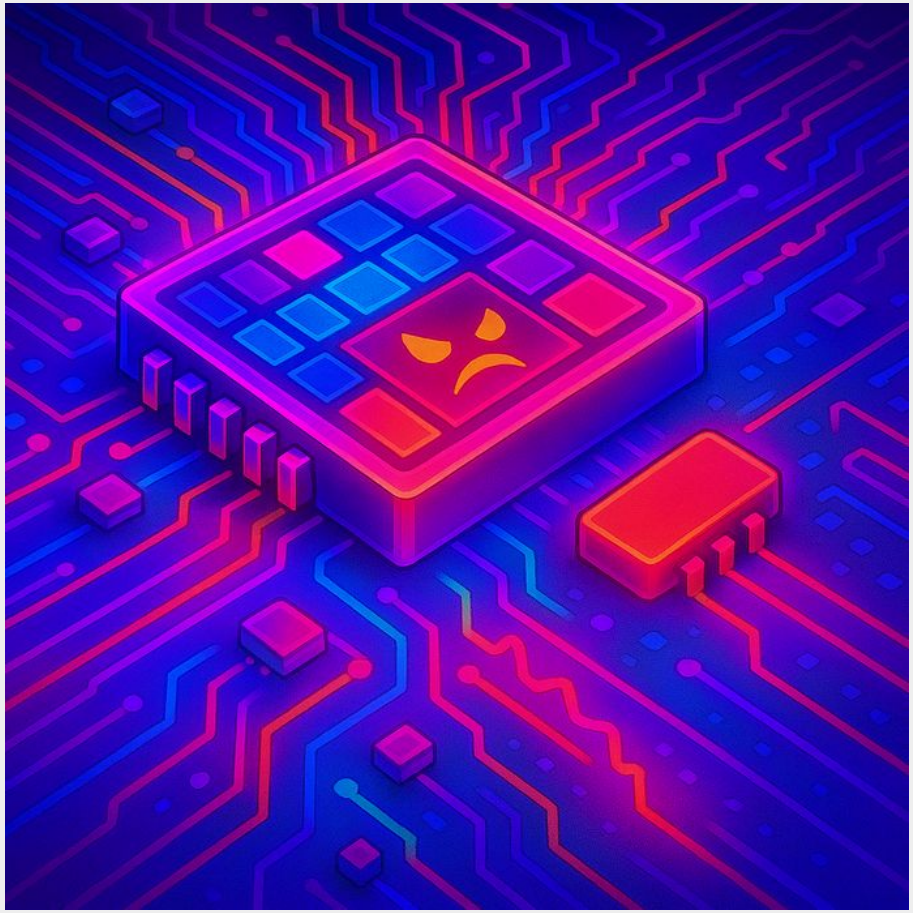
I bet you are by now

Hardware Security
Week 5 - SoC Security

26T1

Hammond Pearce

Slide material acknowledgements to
Jason Blocklove



Security Assurance

- Hardware provides the root of trust for the CIA triad
  - Confidentiality
  - Integrity
  - Availability
- Security assurance refers to the process of verifying that a design meets security objectives
- How can this be achieved once a design has been manufactured and distributed?

Security Objectives

- What is a security objective?
  - Security goal for what to protect
  - Intended response to its attack

Security Objective Examples

Security objectives can vary based on market:

- Data center product
  - low probability of physical attack
  - high probability of remote attack
- IoT device
  - high probability of physical attack
  - high probability of remote attack

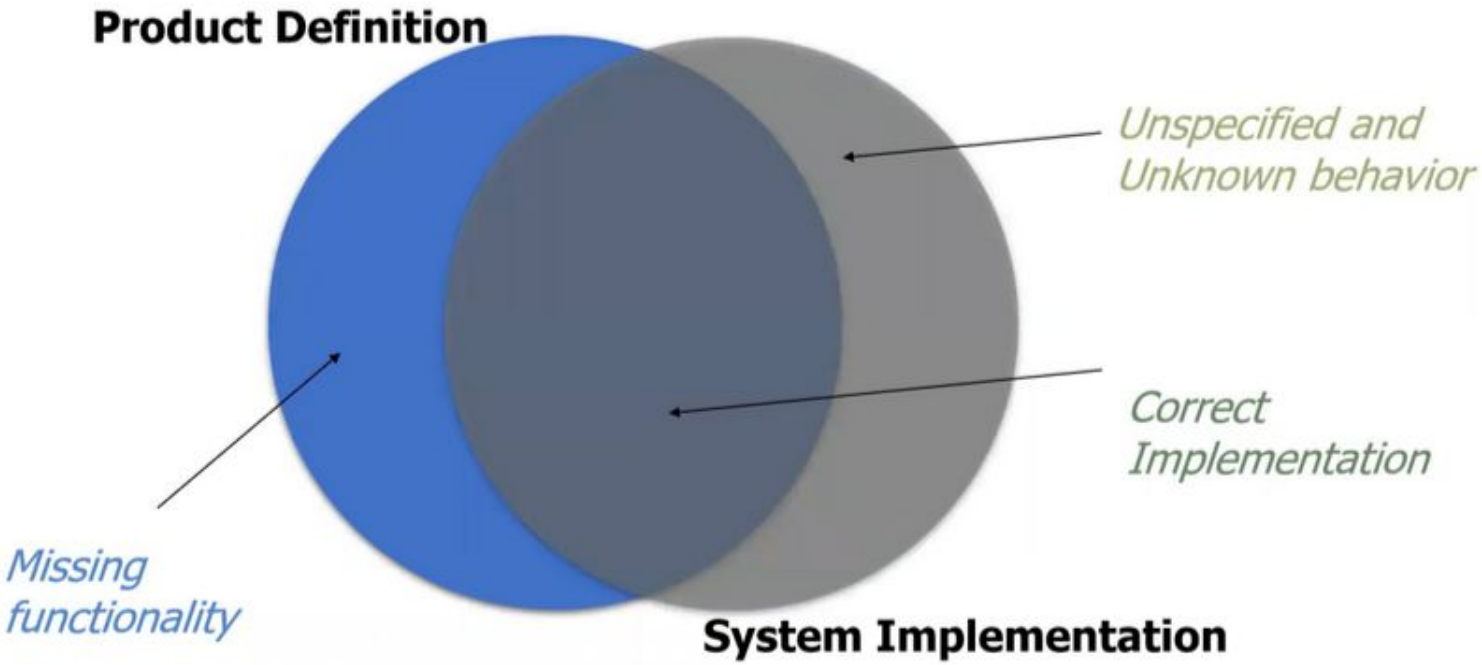
Security Objective Examples

Example: An SoC is being made for low power data transfer applications

- Security goal: the SoC only executes authenticated firmware

- Response: If non-authenticated firmware is loaded the device will enter an error state requiring a power-cycle to reload firmware

Need for Validation



Functional vs. Security Validation

- Functionality testing: Test plan
  - Meets specifications?
  - Documented?
  - Checks anticipated use cases & corner cases
- Security testing: Security plan
  - Determine a threat model
    - Check security targets
    - Anticipate weaknesses
  - Derive security test cases based on threat models
    - Abstract corner cases
    - Protect “impossible” states
    - Side channel analysis

Functional vs. Security Bug

Functional Bug: Detected during normal verification/validation

- Specification violation
- Error giving an incorrect result
- Crashes/failures in normal operation
- Example: Phone touch screen stops working when incorrectly unlocked

Security Bug: Detected when validating for security objectives

- Checks against threat model
- Leverages unspecified operation
- Makes the product vulnerable to attacks
- Example: Spectre & Meltdown

SoC Development Lifecycle

Architecture
- Define functionality
- Hardware Architecture Specifications (HAS)

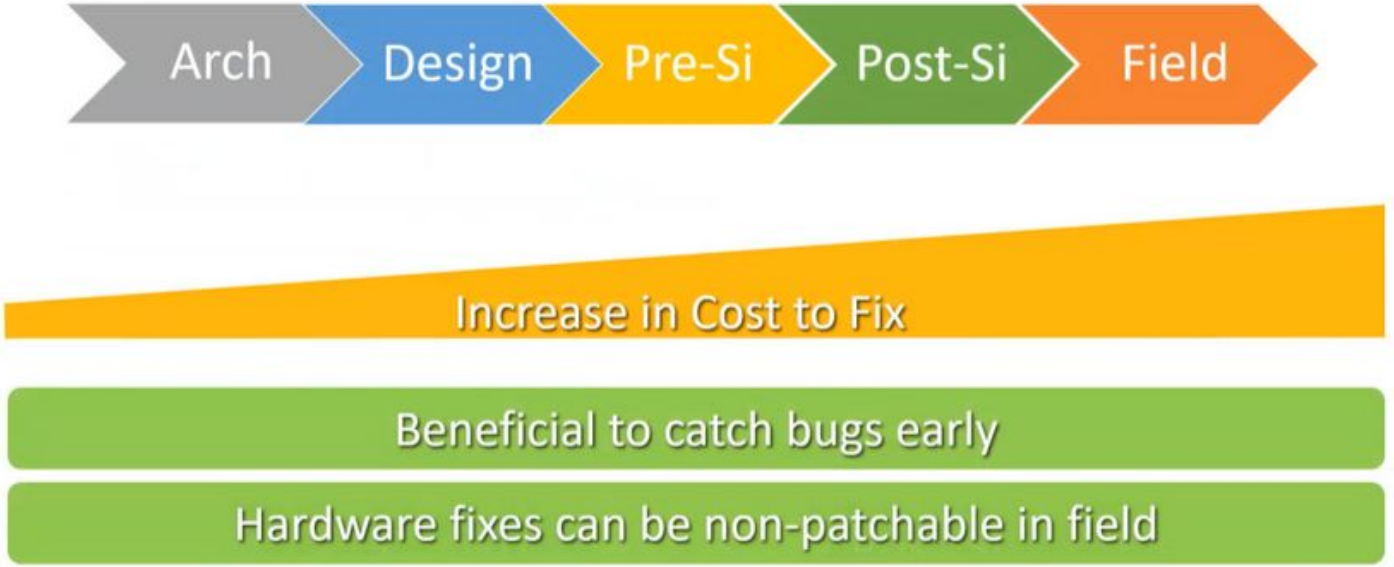
Design
- Implement functionality in hardware
- Microarchitecture Specification (MAS)

Pre-Silicon Verification
- Simulation, emulation, formal verification

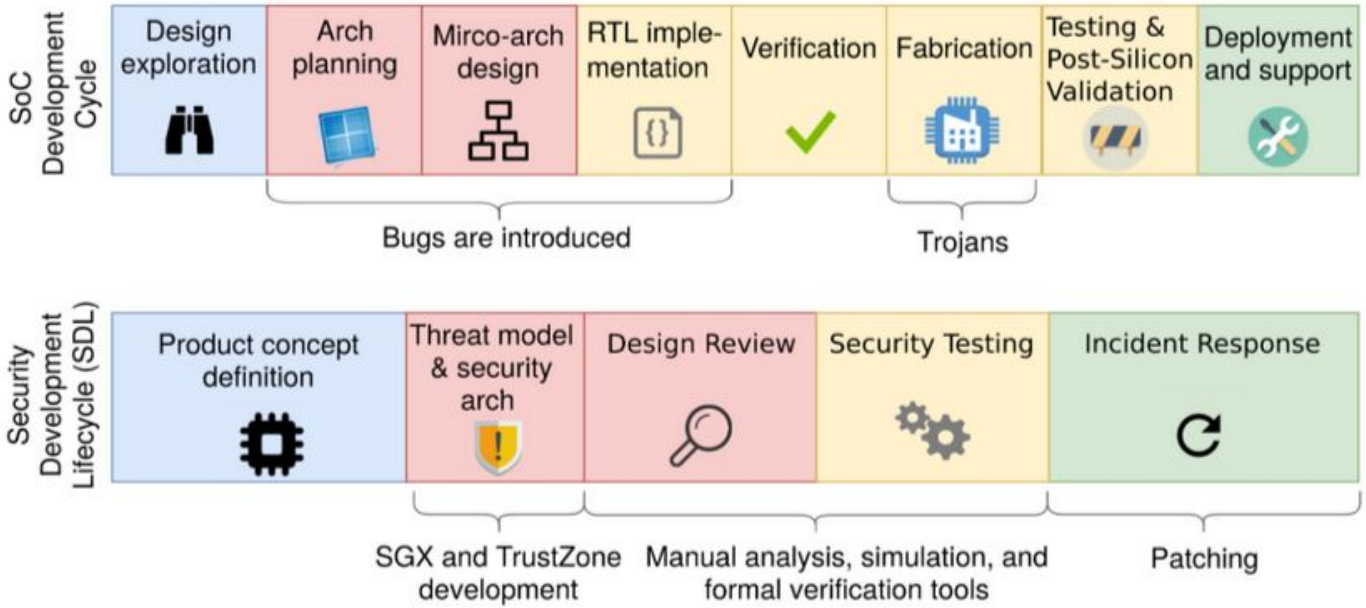
Post-Silicon Verification
- Testing manufactured hardware
- Using hardware debuggers
- If failures are found the chip is fully analyzed

Field Analysis
- Post-mortem
- User reports

SoC Development Cost



Security Development Lifecycle



https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky

Hardware Security Features for SoCs

- Fabric access control
- Control register locks and permissions
- Cryptography hardware
- Memory isolation
- Secure debugging

and their pitfalls:

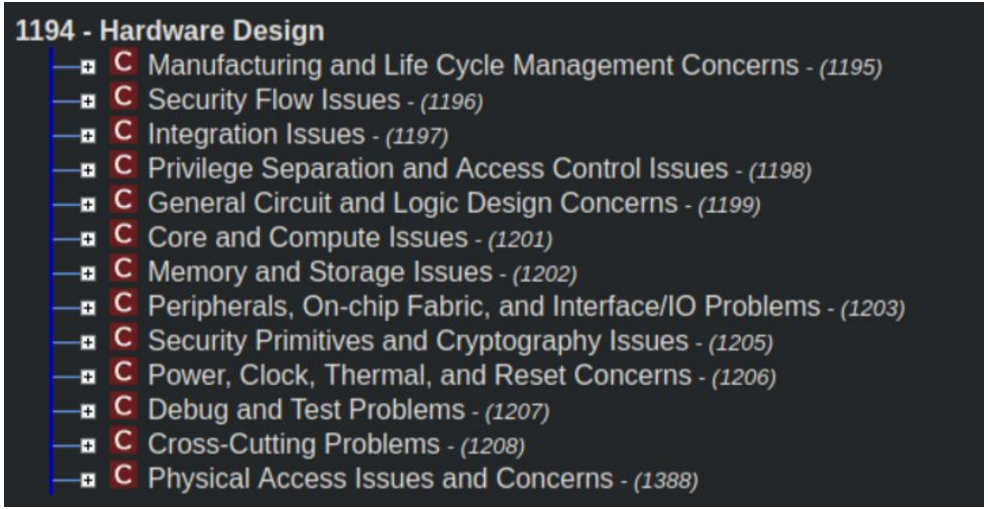
- Improper register attributes
- Register locking (incorrectly unlocking or no locks)
- Memory security
- Crypto hardware implementation bugs
- Debug hardware implementation issues

and their pitfalls:

- Improper register attributes
- Register locking (incorrectly unlocking or no locks)
- Memory security
- Crypto hardware implementation bugs
- Debug hardware implementation issues

Question: Where do the attacks we've looked at so far fit into this list?

Hardware CWEs



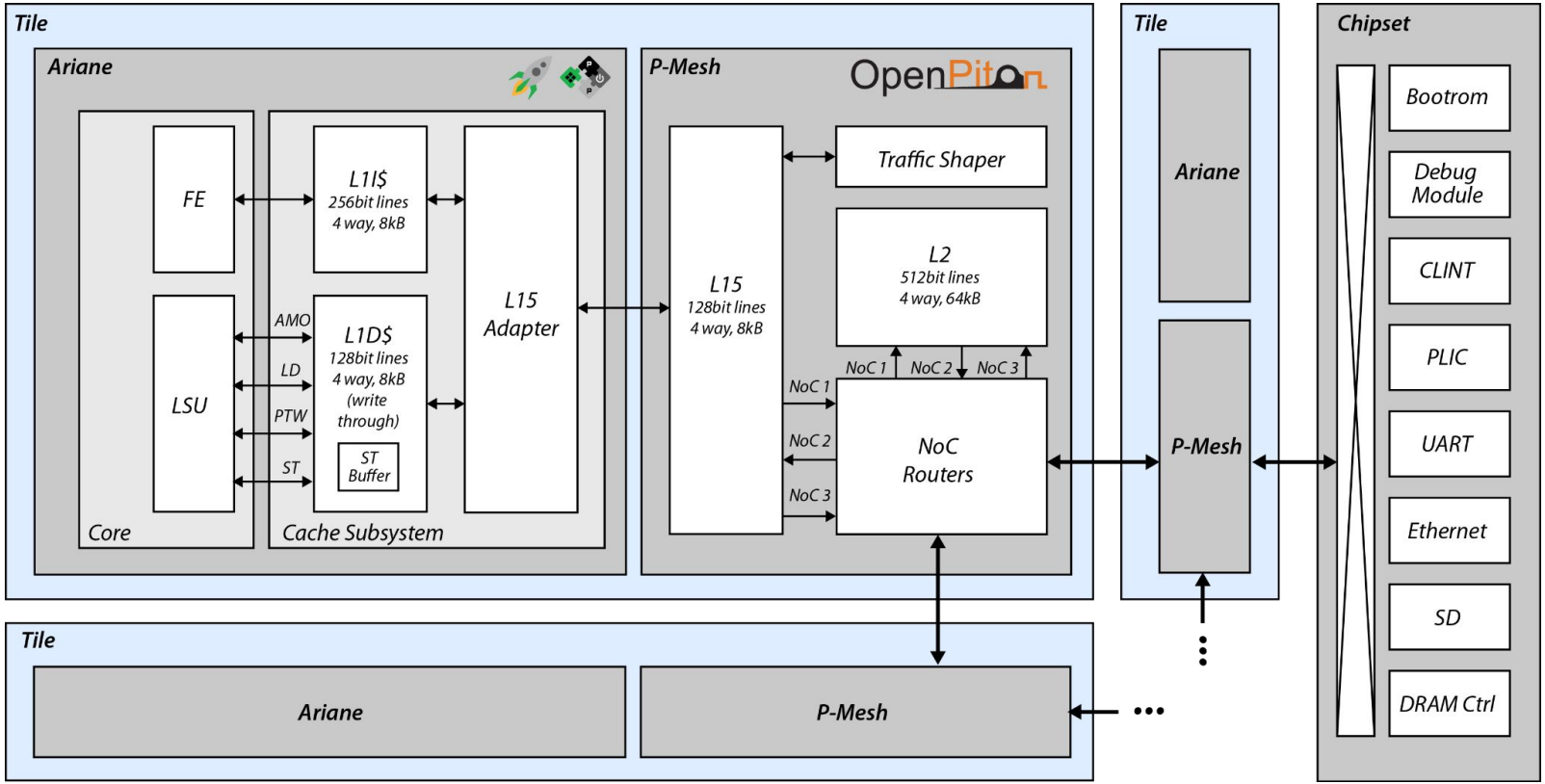
- Hardware Common Weakness Enumeration (CWE)
- Categorises/Taxonomises possible security-relevant faults
- CVE: (Common Vulnerability and Exposure): A specific incident

Categorising Fault Severity

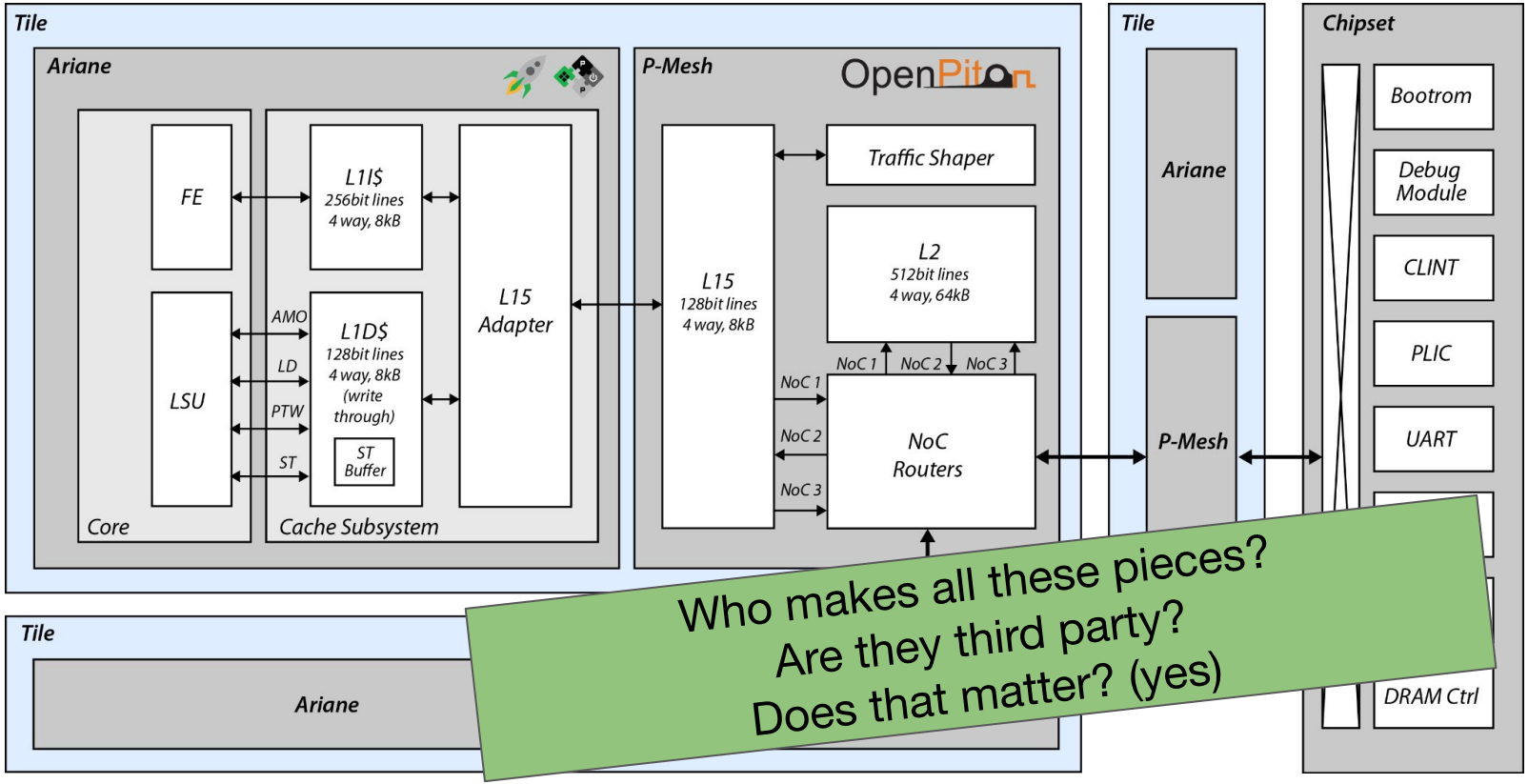
Common Vulnerability Scoring System (CVSS)

https://nvd.nist.gov/vuln-metrics/cvss

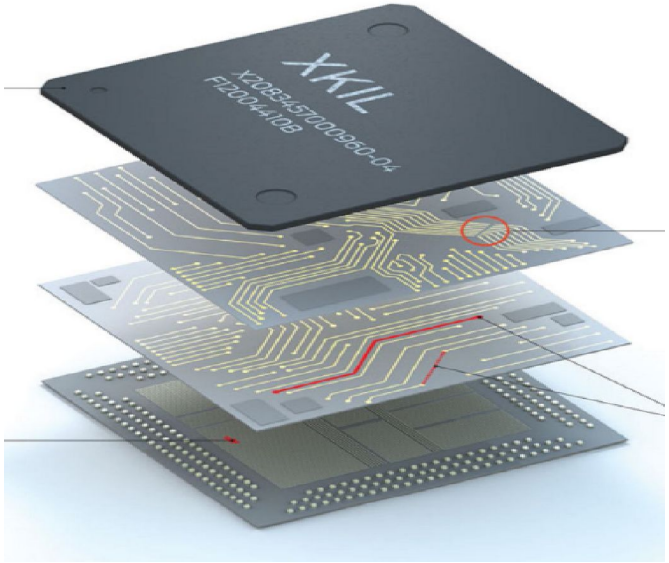
Example: Ariane & OpenPiton



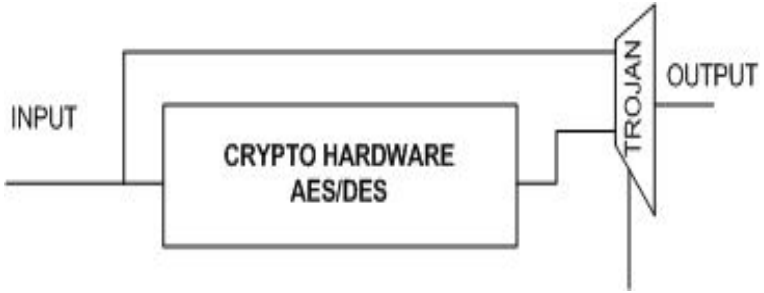
Example: Ariane & OpenPiton



Hardware Trojans: The Kill Switch?



IEEE Spectrum, 2008

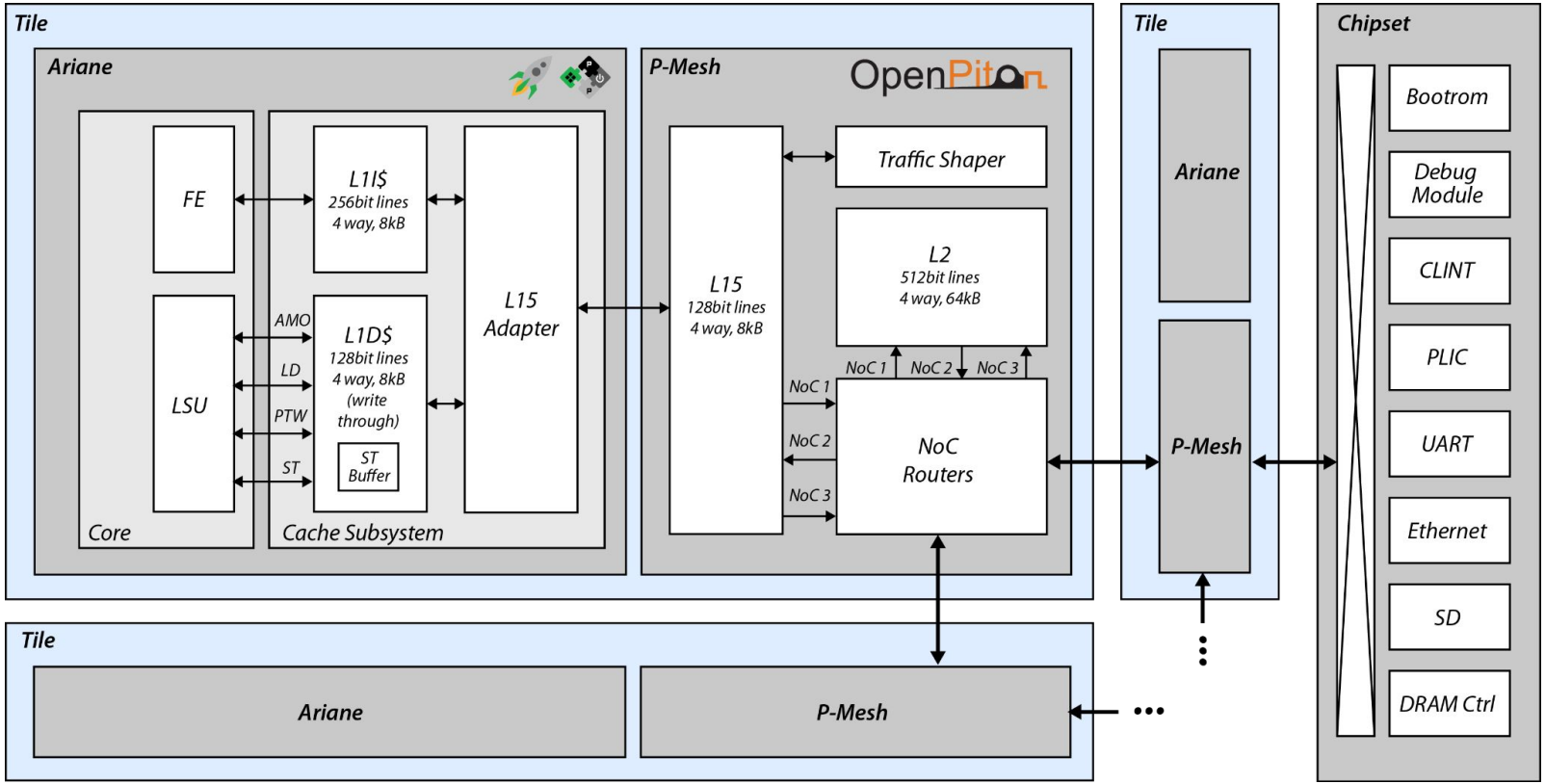


Hardware Trojans can enter via 3PIP

- Risk profile same as at PCB level

- Unintended access / changes to signals, busses, and modules
  - Data leakage (violate confidentiality)
  - Data changes (violate integrity)
  - Deny service (violate availability)

Class exercise: Thinking of Trojans



Hack@DAC: Actually-inserted Trojans ‘21

99 different Hardware Trojans!!

https://github.com/HACK-EVENT/hackatdac21