Fault Attacks

Side-Channel Security

Rishub Nagpal

June 13, 2024

IAIK – Graz University of Technology

Quick note on Master's Project/Thesis

If you found (parts) of this lecture interesting, consider doing a master project/thesis with us!

Some topics I offer:

- Single-Trace Side Channel Attacks (Like task 2, but more detail)
- Masking Countermeasures for Software and Hardware
- Machine Learning for SCA
- Implementing SCA tools efficiently in Rust
- Attacking Post-Quantum crypto on real devices

Overview

Recap

Differential Fault Attacks

Statistical Fault Attacks

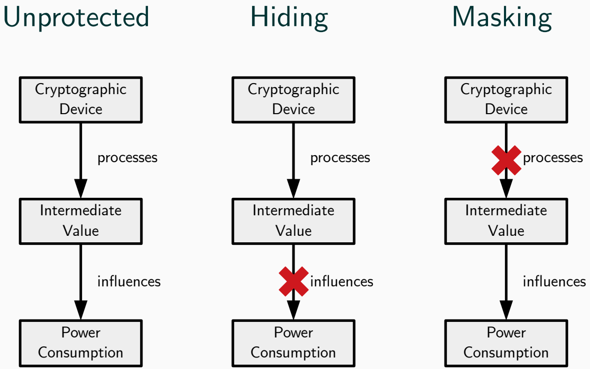
Countermeasures

Breaking Countermeasures Again

Recap

Last Weeks...

- Power analysis attacks
- Countermeasures
  - Hiding (Shuffling)
  - Masking



Unprotected

Cryptographic Device → processes → Intermediate Value → influences → Power Consumption

Hiding

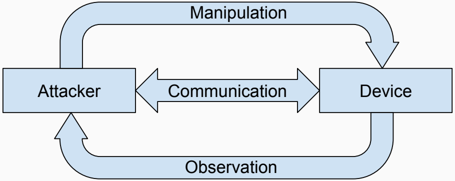
Cryptographic Device → processes → Intermediate Value → influences → Power Consumption

Masking

Cryptographic Device → processes → Intermediate Value → influences → Power Consumption

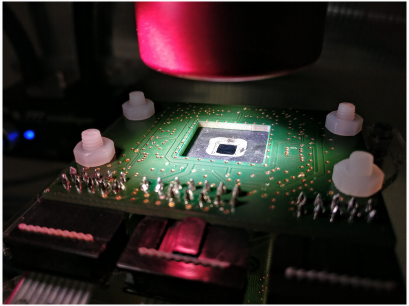
Physical-Attack Scenario

- Attacker has (legitimate) access to device
- Thus far: Passive attacks (and countermeasures)
- But the attacker can do much more. . .



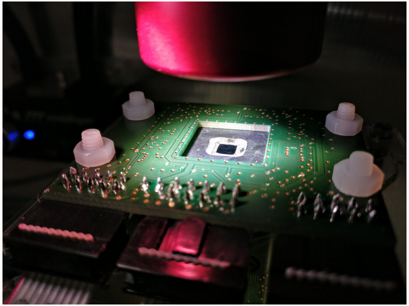
Fault Attacks

- Induce fault in computation: Erroneous result
    - Transient faults: Only current computation (gone after reset, at the latest)



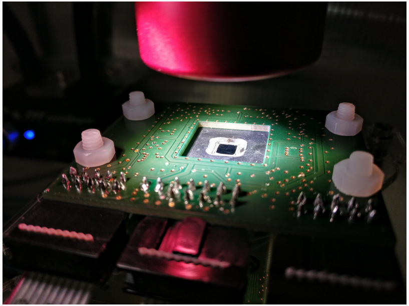
Fault Attacks

- Induce fault in computation: Erroneous result
  - Transient faults: Only current computation (gone after reset, at the latest)
- Fault injection techniques
  - Spike/glitch attacks (clock, Vdd, IO, . . .)
  - Laser, BBI
  - Rowhammer, Plundervolt
  - . . .



Fault Attacks

- Induce fault in computation: Erroneous result
  - Transient faults: Only current computation (gone after reset, at the latest)
- Fault injection techniques
  - Spike/glitch attacks (clock, Vdd, IO, ...)
  - Laser, BBI
  - Rowhammer, Plundervolt
  - ...
- Effects
  - Instruction skip
  - Data corruption
  - ...



Remember?

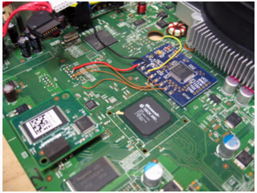
- PayTV (early 2000s)
  - Pirated cards bricked via remote firmware update
  - Inserted infinite loop, otherwise unchanged
  - Solution: Glitching! Increment IP, but no jmp
  - “Unlooper” device

```
// startup loop:
jmp loop;
// continue to bootloader
```

Remember?

- PayTV (early 2000s)
  - Pirated cards bricked via remote firmware update
  - Inserted infinite loop, otherwise unchanged
  - Solution: Glitching! Increment IP, but no jmp
  - "Unlooper" device
- Gaming devices
  - Xbox360 reset hack
  - Voltage glitching on reset line
  - Execute untrusted code (modified firmware)

```
// startup loop:
jmp loop;
// continue to bootloader
```



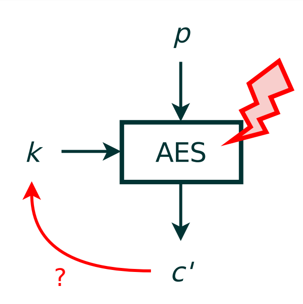
Our Scenario

- Attack cryptographic implementations
- We want to get the key
- Fault injection alone (mostly) does not leak the key
  → More work is needed

Differential Fault Attacks

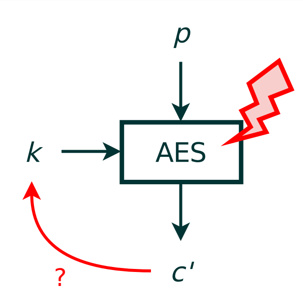
Basic Idea

- Inject fault during AES encryption
  - Get: Faulty ciphertext c'
  - Want: Key
  - → Faulting alone is only half the game!

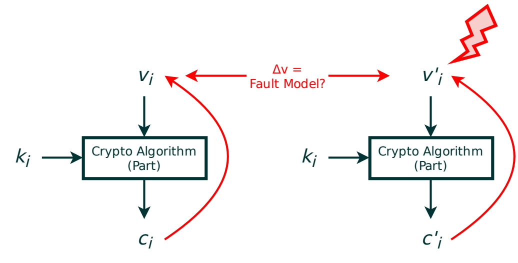


Basic Idea

- Inject fault during AES encryption
  - Get: Faulty ciphertext c'
  - Want: Key
  - → Faulting alone is only half the game!
- Idea: Compare correct and faulty ciphertext
  - Encrypt same plaintext twice, once with a fault
  - Use difference in ciphertext to recover the key
  - → Differential Fault Attack

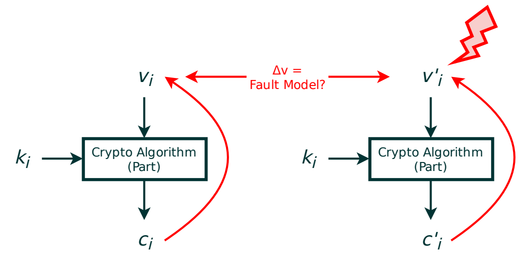


Differential Fault Attacks – Basic Principle



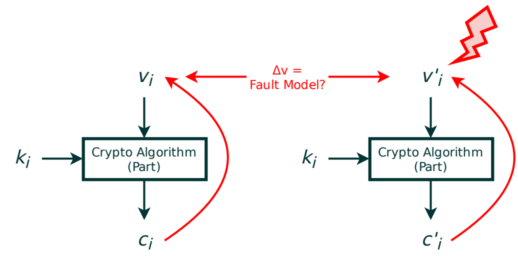
- Pick an intermediate v
    - v is combined with a small part of the last round key

Differential Fault Attacks – Basic Principle



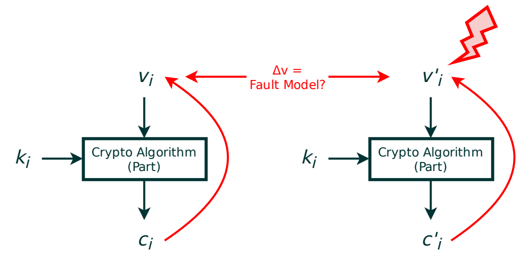
- Pick an intermediate v
  - v is combined with a small part of the last round key
- 2 invocations with same p, once with fault in v
  - Usually we don't know exact effect
  - Could be flipping 1 bit/byte
  - Could be randomization of 1 bit/byte

Differential Fault Attacks – Basic Principle cont.



- Enumerate possible subkey values
  - Compute backwards for each guess
  - Check if XOR-difference = fault model
  - Wrong guess: "randomized" v and Δv

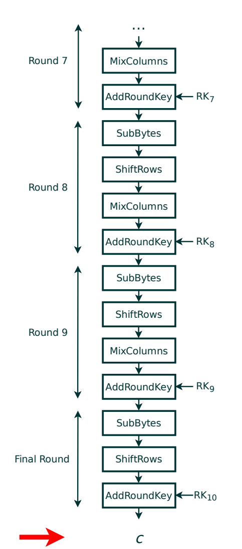
Differential Fault Attacks – Basic Principle cont.



- Enumerate possible subkey values
    - Compute backwards for each guess
    - Check if XOR-difference = fault model
    - Wrong guess: “randomized” v and Δv
- Remember: AES key schedule is invertible
    - If it were not: Attack decryption or attack round after round

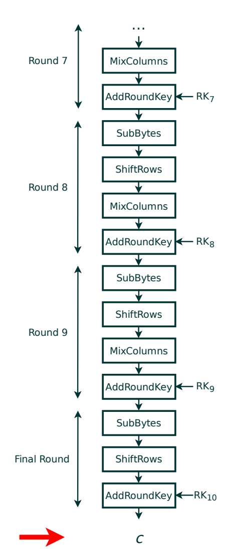
Differential Fault Attacks on AES

- Faulting Ciphertext?



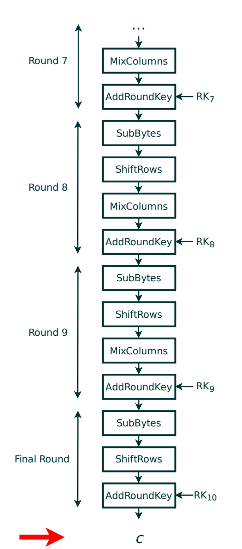
Differential Fault Attacks on AES

- Faulting Ciphertext?
- No!



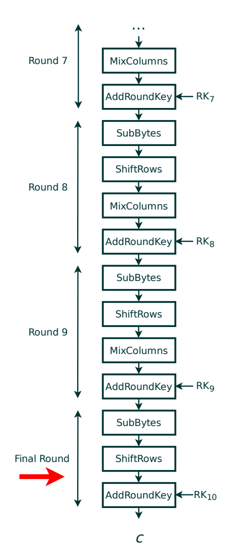
Differential Fault Attacks on AES

- Faulting Ciphertext?
- No!
- Δc does not depend on a key



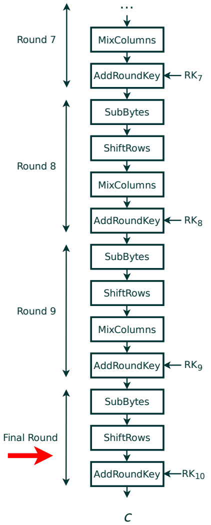
Differential Fault Attacks on AES

- Faulting before AddRoundKey10?
  - . . . depends on type of fault



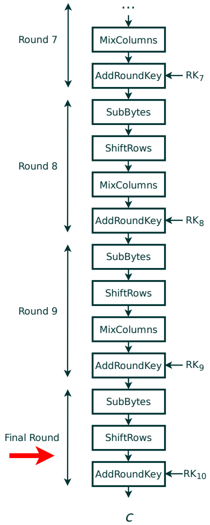
Differential Fault Attacks on AES

- Faulting before AddRoundKey10?
  - . . . depends on type of fault
- Fault model 1: Stuck-at known (can set v to a specific value)
  - Attack possible!
  - Example: v stuck at 0 → $c = v \oplus k = k$



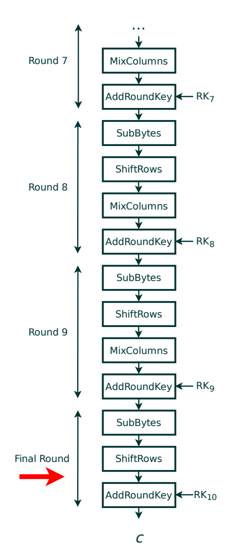
Differential Fault Attacks on AES

- Faulting before AddRoundKey10?
  - . . . depends on type of fault
- Fault model 1: Stuck-at known (can set v to a specific value)
  - Attack possible!
  - Example: v stuck at 0 $\rightarrow c = v \oplus k = k$
- Problem: Stuck-at-known hard to do (reliably)
  - Easier: Random flips, stuck-at-unknown, ...



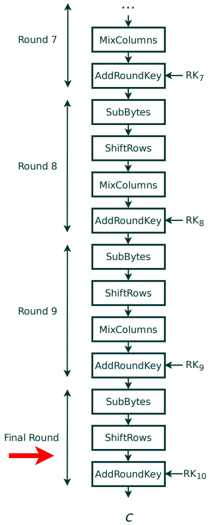
Differential Fault Attacks on AES

- Faulting before AddRoundKey10?
    - . . . depends on type of fault



Differential Fault Attacks on AES

- Faulting before AddRoundKey10?
  - . . . depends on type of fault
- Fault model 2: Random flips

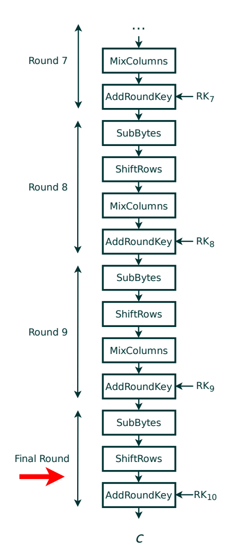


Differential Fault Attacks on AES

- Faulting before AddRoundKey10?
  - . . . depends on type of fault
- Fault model 2: Random flips
  - No attack possible!
  - Fault propagates through XOR → Δc does not depend on the key

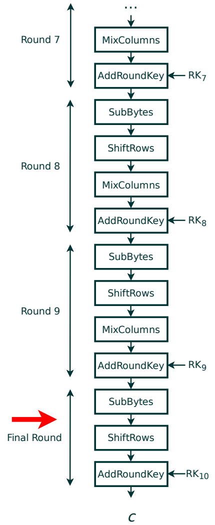
    $c = v \oplus k$

    $c' = (v \oplus \Delta v) \oplus k = c \oplus \Delta v$



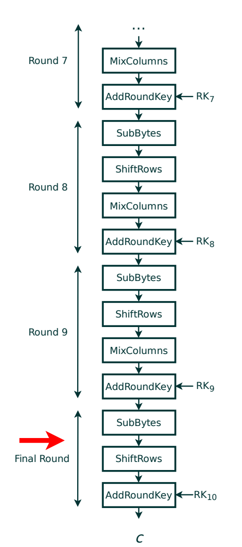
Differential Fault Attacks on AES

- Faulting before ShiftRows10?



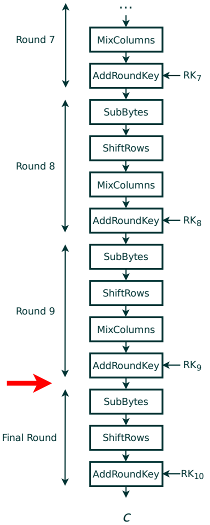
Differential Fault Attacks on AES

- Faulting before ShiftRows10?
- Same situation as for AddRoundKey
  - Attack possible
  - ShiftRows just rearranges bytes



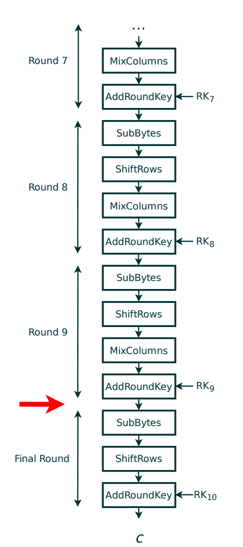
Differential Fault Attacks on AES

- Faulting before SubBytes10?
  - . . . depends on fault type



Differential Fault Attacks on AES

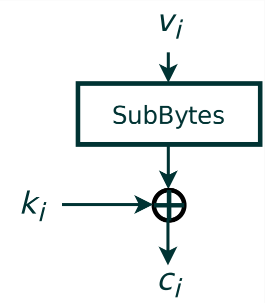
- Faulting before SubBytes10?
    - . . . depends on fault type
- Fault Model 1: Flip 1 bit
    - Attack possible
    - . . . but hard to achieve single flipping bit (precise Laser)



Single Bit Fault before SubBytes - Example

- Correct output = 1a, faulty output = 99

```
                      k =  0  1  2  3  4  5  6  7  8 ...
--------------------------------------------------------
C   = 1a : S^-1(C  xor k):
C’  = 99 : S^-1(C’ xor k):
```

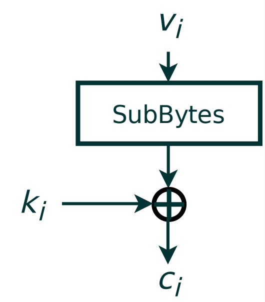


Single Bit Fault before SubBytes - Example

- Correct output = 1a, faulty output = 99

```
                      k =  0  1  2  3  4  5  6  7  8 ...
-----------------------------------------------------
C  = 1a : S^-1(C  xor k): 43 44 34 8e e9 cb c4 de 39 ...
C' = 99 : S^-1(C' xor k): f9 e2 e8 37 75 1c 6e df ac ...
```

- Only few keys have this property: Filter them!

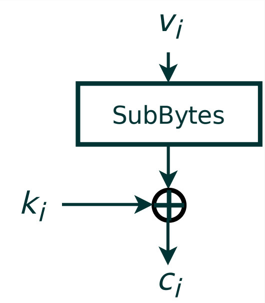


Single Bit Fault before SubBytes - Example

- Correct output = 1a, faulty output = 99

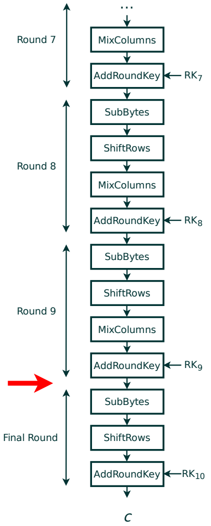
```
                      k =  0  1  2  3  4  5  6  7  8 ...
-----------------------------------------------------
C  = 1a : S^-1(C  xor k): 43 44 34 8e e9 cb c4 de 39 ...
C' = 99 : S^-1(C' xor k): f9 e2 e8 37 75 1c 6e df ac ...
```

- Only few keys have this property: Filter them!
- Use another c/c' pair to get down to 1 key



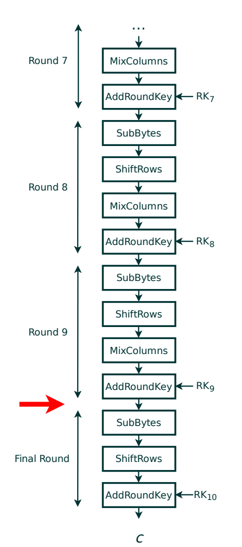
Differential Fault Attacks on AES

- Faulting before SubBytes10?
    - . . . depends on fault type
- Fault Model 2: Random byte fault (unknown)



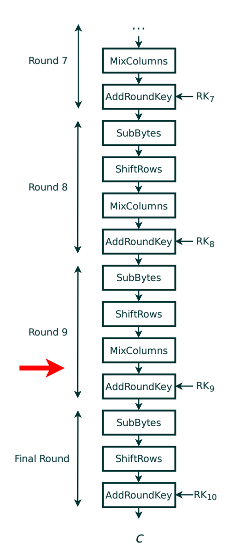
Differential Fault Attacks on AES

- Faulting before SubBytes10?
  - . . . depends on fault type
- Fault Model 2: Random byte fault (unknown)
  - Much easier to achieve (on an 8-bit implementation)
  - . . . but no attack possible



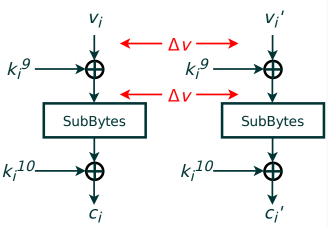
Differential Fault Attacks on AES

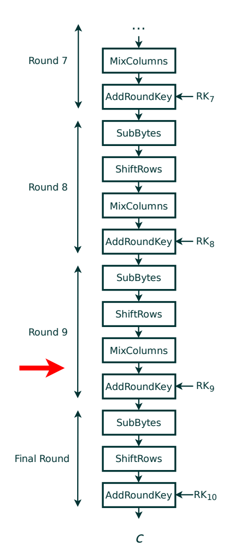
- Faulting before AddRoundKey9?



Differential Fault Attacks on AES

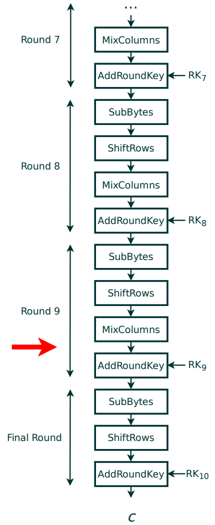
- Faulting before AddRoundKey9?
- Important observation: $\Delta v = v \oplus v' = (v \oplus k) \oplus (v' \oplus k)$





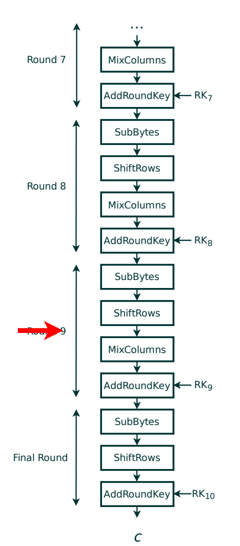
Differential Fault Attacks on AES

- Faulting before AddRoundKey9?
  - Attack possible
  - Exactly the same as previously (RK_9 cancels out)



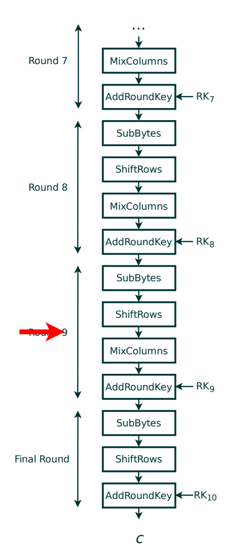
Differential Fault Attacks on AES

- Faulting before MixColumns9?



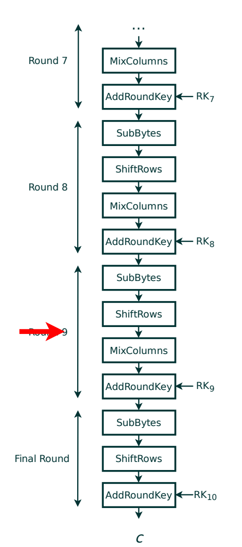
Differential Fault Attacks on AES

- Faulting before MixColumns9?
- Fault Model: Random byte fault (unknown)
  - Attack possible!
  - Comparably easy to achieve
  - Basis for Piret’s attack



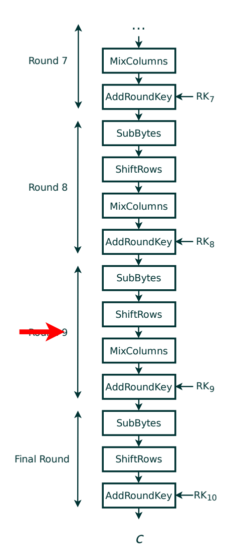
Fault Propagation of MixColumns

- Model byte fault using (unknown) difference Δ: $v_1' = v_1 \oplus \Delta$



Fault Propagation of MixColumns

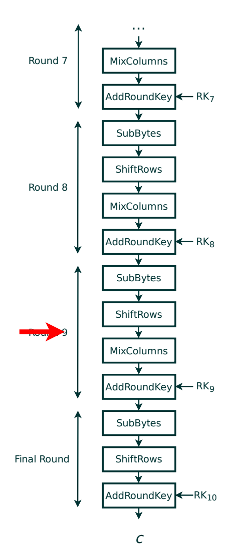
- Model byte fault using (unknown) difference Δ: $v_1' = v_1 \oplus \Delta$
- MixColumns: Linear operation, 4 byte input:
  $\text{MixColumns}([v_1 \oplus \Delta, v_2, v_3, v_4]) =$
  $\text{MixColumns}([v_1 \qquad , v_2, v_3, v_4]) \oplus \text{MixColumns}([\Delta, 0, 0, 0])$



Fault Propagation of MixColumns

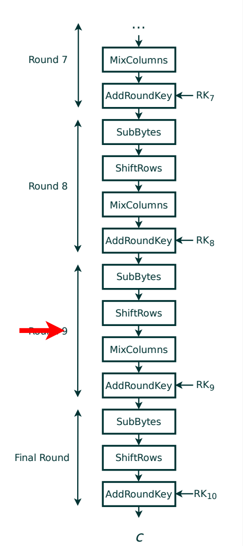
- Model byte fault using (unknown) difference Δ: $v_1' = v_1 \oplus \Delta$
- MixColumns: Linear operation, 4 byte input:

  $$\text{MixColumns}([v_1 \oplus \Delta, v_2, v_3, v_4]) =$$
  $$\text{MixColumns}([v_1 \qquad , v_2, v_3, v_4]) \oplus \text{MixColumns}([\Delta, 0, 0, 0])$$
- Observe: 4 byte input, but only 1 is active in $\text{MixColumns}([\Delta, 0, 0, 0])$
  - Only 255 possible MixColumns outputs!



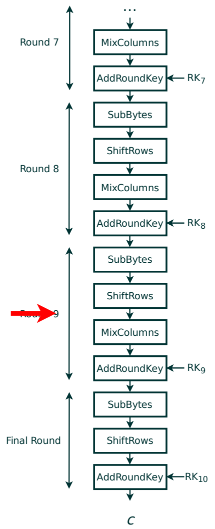
Fault Propagation of MixColumns

- Model byte fault using (unknown) difference Δ: $v_1' = v_1 \oplus \Delta$
- MixColumns: Linear operation, 4 byte input:
  $\text{MixColumns}([v_1 \oplus \Delta, v_2, v_3, v_4]) =$
  $\text{MixColumns}([v_1 \quad\quad , v_2, v_3, v_4]) \oplus \text{MixColumns}([\Delta, 0, 0, 0])$
- Observe: 4 byte input, but only 1 is active in $\text{MixColumns}([\Delta, 0, 0, 0])$
  - Only 255 possible MixColumns outputs!
- Other 3 bytes can be faulted as well (but only one at a time)
  - $\text{MixColumns}([\Delta, 0, 0, 0])$
  - $\text{MixColumns}([0, \Delta, 0, 0])$
  - $\text{MixColumns}([0, 0, \Delta, 0])$
  - $\text{MixColumns}([0, 0, 0, \Delta])$



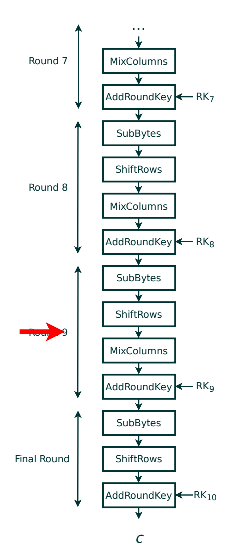
Fault Propagation of MixColumns

- Model byte fault using (unknown) difference Δ: $v_1' = v_1 \oplus \Delta$
- MixColumns: Linear operation, 4 byte input:
  MixColumns($[v_1 \oplus \Delta, v_2, v_3, v_4]$) $=$
  MixColumns($[v_1 \quad\quad , v_2, v_3, v_4]$) $\oplus$ MixColumns($[\Delta, 0, 0, 0]$)
- Observe: 4 byte input, but only 1 is active in MixColumns($[\Delta, 0, 0, 0]$)
  - Only 255 possible MixColumns outputs!
- Other 3 bytes can be faulted as well (but only one at a time)
  - MixColumns($[\Delta, 0, 0, 0]$)
  - MixColumns($[0, \Delta, 0, 0]$)
  - MixColumns($[0, 0, \Delta, 0]$)
  - MixColumns($[0, 0, 0, \Delta]$)
- $4 \times 255 = 1020$ possible MixColumns outputs
  - Precompute all of them



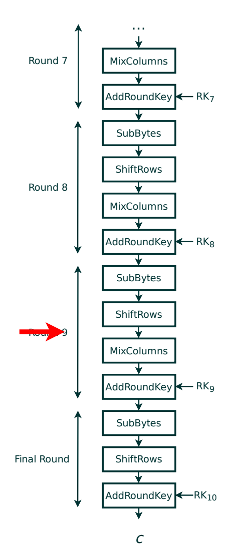
Attack Principle

- Fault random byte before MixColumns9
  - Determine affected column by checking output difference (only 4 ciphertext bytes will differ, check which ones)



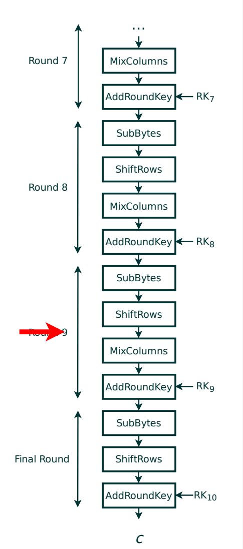
Attack Principle

- Fault random byte before MixColumns9
  - Determine affected column by checking output difference (only 4 ciphertext bytes will differ, check which ones)
- Enumerate 2^{32} combinations of affected RK_{10}
  - Compute back to output of MixColumns9 (for both, valid c and faulty c')
  - Test if difference is in precomputed list
    → If yes then keep key candidate



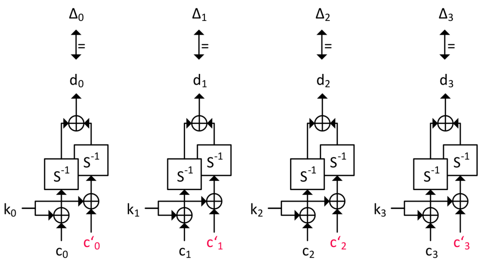
Attack Principle

- Fault random byte before MixColumns9
  - Determine affected column by checking output difference (only 4 ciphertext bytes will differ, check which ones)
- Enumerate 2^{32} combinations of affected RK_{10}
  - Compute back to output of MixColumns9 (for both, valid c and faulty c')
  - Test if difference is in precomputed list
    → If yes then keep key candidate
- Problem: We don't want to try 2^{32} keys . . .



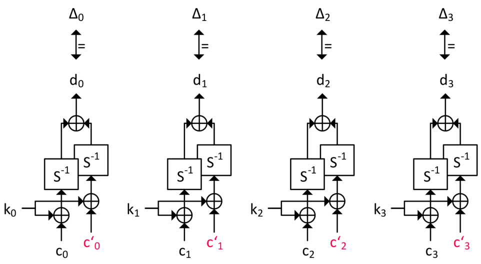
Efficient Implementation of the Attack

1. Compute all 1020 possible MC output differences
   - MC(1. . .255, 0, 0, 0), MC(0, 1. . .255, 0, 0), . . .



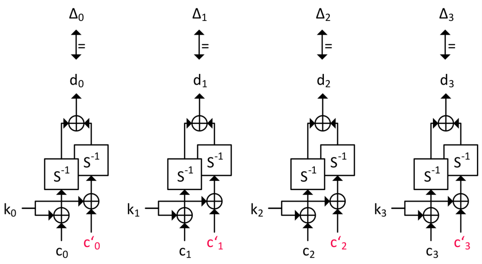
Efficient Implementation of the Attack

1. Compute all 1020 possible MC output differences
   - MC(1. . .255, 0, 0, 0), MC(0, 1. . .255, 0, 0), . . .
2. Predict differences for each key byte individually
   - 4 positions × 256 values = 1024 combinations
   - Precompute them once



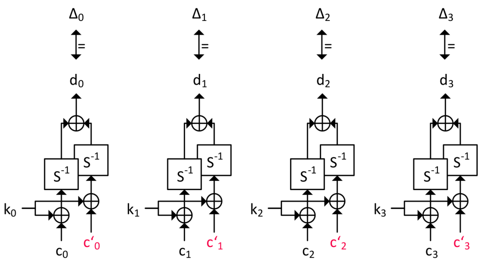
Efficient Implementation of the Attack

1. Compute all 1020 possible MC output differences
   - MC(1. . .255, 0, 0, 0), MC(0, 1. . .255, 0, 0), . . .
2. Predict differences for each key byte individually
   - 4 positions × 256 values = 1024 combinations
   - Precompute them once
3. Loop over possible differences:
   - For each difference $(\Delta_0, \Delta_1, \Delta_2, \Delta_3)$:
   - Add all combinations of key bytes where $(\Delta_0 = d_0, \Delta_1 = d_1, \ldots)$ to a list of potential keys



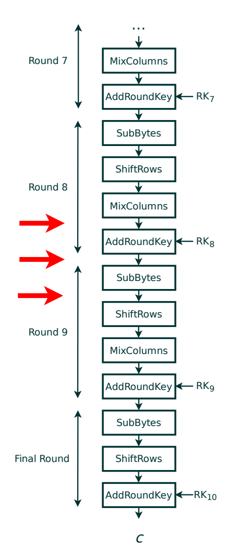
Efficient Implementation of the Attack

1. Compute all 1020 possible MC output differences
   - MC(1. . .255, 0, 0, 0), MC(0, 1. . .255, 0, 0), . . .
2. Predict differences for each key byte individually
   - 4 positions × 256 values = 1024 combinations
   - Precompute them once
3. Loop over possible differences:
   - For each difference $(\Delta_0, \Delta_1, \Delta_2, \Delta_3)$:
   - Add all combinations of key bytes where $(\Delta_0 = d_0, \Delta_1 = d_1, \ldots)$ to a list of potential keys
4. Use second faulty/correct pair
   - Loop over remaining key candidates from previous step
   - For each: Test if predicted difference is in precomputed list



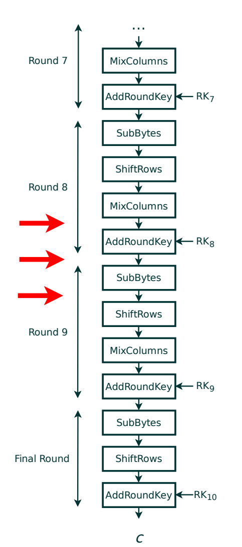
Attack Principle

- Faulting between AddRoundKey8 - SubBytes9?



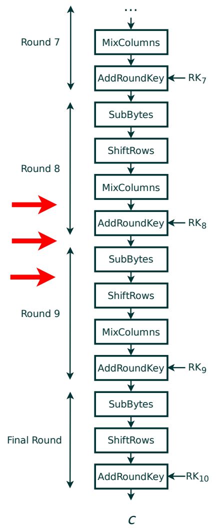
Attack Principle

- Faulting between AddRoundKey8 - SubBytes9?
- Fault model: Random fault in 1 byte
  - Attack possible!
  - Leads to exactly the same scenario as previously (1 byte in MixColumns9 is incorrect)



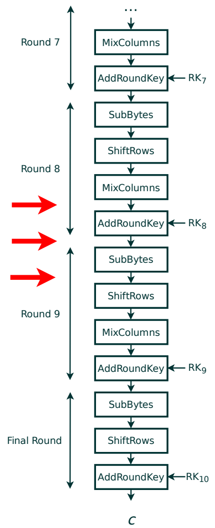
Attack Principle

- Faulting between AddRoundKey8 - SubBytes9?
- Fault model: Random fault in 1 byte
  - Attack possible!
  - Leads to exactly the same scenario as previously (1 byte in MixColumns9 is incorrect)
- ⇒ Powerful attack that requires at least 8 faults (at 4 locations) for full key recovery



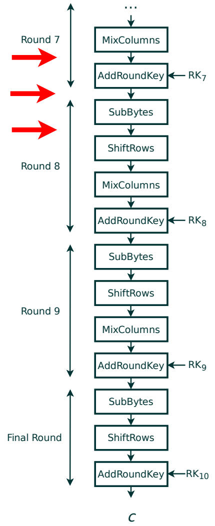
Attack Principle

- Faulting between AddRoundKey8 - SubBytes9?
- Fault model: Random fault in 1 byte
  - Attack possible!
  - Leads to exactly the same scenario as previously (1 byte in MixColumns9 is incorrect)
- ⇒ Powerful attack that requires at least 8 faults (at 4 locations) for full key recovery
- But can we do even better?



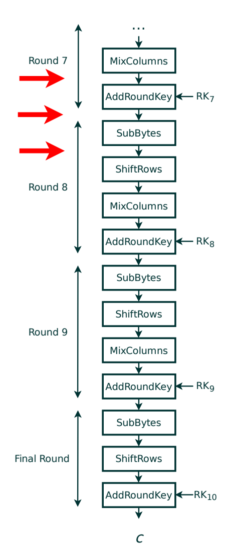
Attack Principle

- Faulting between AddRoundKey7 - MixColumns8



Attack Principle

- Faulting between AddRoundKey7 - MixColumns8
- Fault model: Random fault in 1 byte
  - Attack possible!
  - Observation: faulting 1 byte
    → All 4 bytes in column affected later
  - ShiftRows9 distributes the 4 bytes to 4 different columns



Example

$$\begin{pmatrix} \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{MC8}} \begin{pmatrix} \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{SR9}} \begin{pmatrix} \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta \\ 0 & 0 & \Delta & 0 \\ 0 & \Delta & 0 & 0 \end{pmatrix}$$

- Single-byte difference in each column with just 1 fault!

Example

$$\begin{pmatrix} \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{MC8}} \begin{pmatrix} \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{SR9}} \begin{pmatrix} \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta \\ 0 & 0 & \Delta & 0 \\ 0 & \Delta & 0 & 0 \end{pmatrix}$$

- Single-byte difference in each column with just 1 fault!
- Full key recovery with just 2 faults (by performing Priet's attack $4\times$)

Example

$$\begin{pmatrix} \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{MC8}} \begin{pmatrix} \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \\ \Delta & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{SR9}} \begin{pmatrix} \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta \\ 0 & 0 & \Delta & 0 \\ 0 & \Delta & 0 & 0 \end{pmatrix}$$

- Single-byte difference in each column with just 1 fault!
- Full key recovery with just 2 faults (by performing Priet's attack $4\times$)
- Problem: harder to detect if fault injection is exploitable
  - Before: 4 bytes different → likely exploitable
  - Now: All bytes different, did we really hit single byte before MC8 or something else?

Link to EX2

- Task 4: Piret's Attack

Link to EX2

- Task 4: Piret's Attack
- Task 4.1: Develop the attack using simulated faults
    - Use "Excel implementation" of AES for that
    - Also very useful for debugging

Link to EX2

- Task 4: Piret's Attack
- Task 4.1: Develop the attack using simulated faults
  - Use "Excel implementation" of AES for that
  - Also very useful for debugging
- Task 4.2: Repeat attack with the CW board (voltage glitches)

Link to EX2

- Task 4: Piret's Attack
- Task 4.1: Develop the attack using simulated faults
  - Use "Excel implementation" of AES for that
  - Also very useful for debugging
- Task 4.2: Repeat attack with the CW board (voltage glitches)
- Task 4.3: Implement and test a countermeasure against DFA

Link to EX2

- Task 4: Piret's Attack
- Task 4.1: Develop the attack using simulated faults
  - Use "Excel implementation" of AES for that
  - Also very useful for debugging
- Task 4.2: Repeat attack with the CW board (voltage glitches)
- Task 4.3: Implement and test a countermeasure against DFA
- In all cases: Recovering 4 bytes of the last roundkey is sufficient

Statistical Fault Attacks

Statistical Fault Attacks

- Exploit faulty ciphertexts only

Statistical Fault Attacks

- Exploit faulty ciphertexts only
- Plaintexts can be unknown but need to vary
    - “Opposite” requirement compared to differential attacks

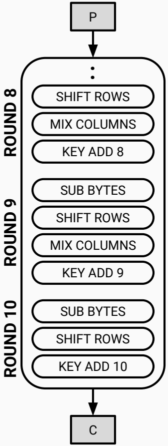
Statistical Fault Attacks

- Exploit faulty ciphertexts only
- Plaintexts can be unknown but need to vary
    - “Opposite” requirement compared to differential attacks
- Usually need more than 2 faulted encryptions

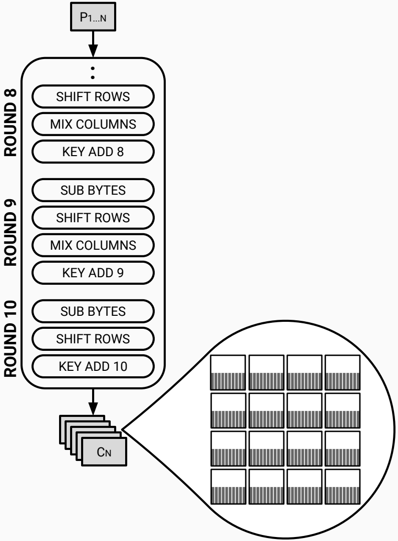
Statistical Fault Attacks

- Exploit faulty ciphertexts only
- Plaintexts can be unknown but need to vary
  - “Opposite” requirement compared to differential attacks
- Usually need more than 2 faulted encryptions
- Key recovery exploits statistical distributions of state bytes (in contrast to differences)

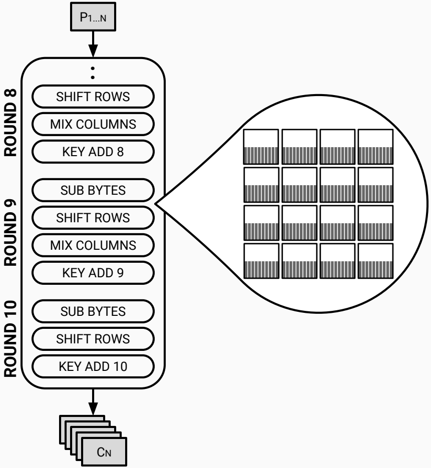
Statistical Fault Attacks on AES in Pictures



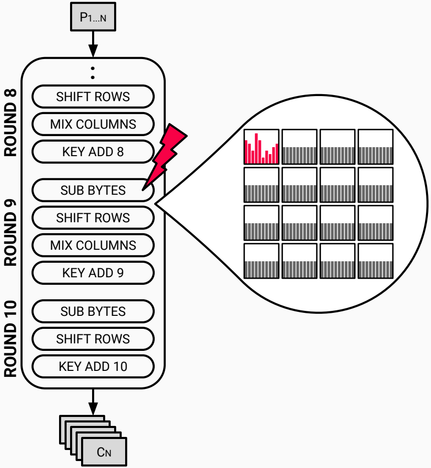
Statistical Fault Attacks on AES in Pictures



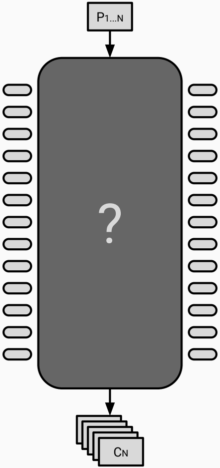
Statistical Fault Attacks on AES in Pictures



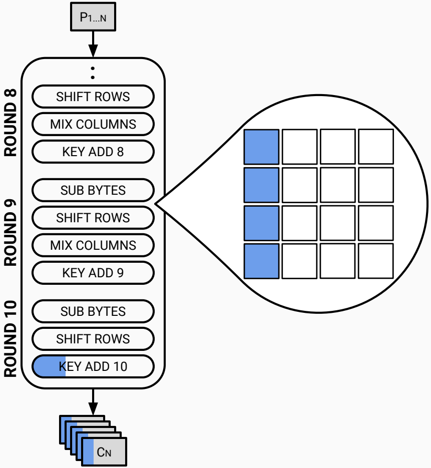
Statistical Fault Attacks on AES in Pictures



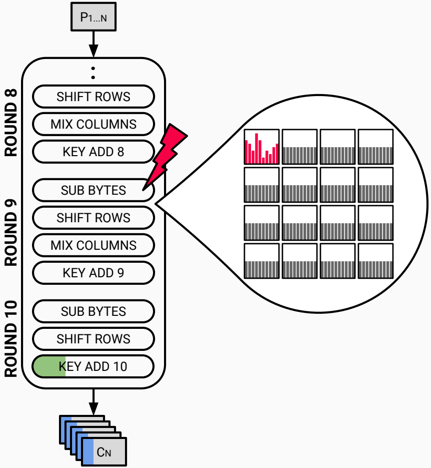
Statistical Fault Attacks on AES in Pictures



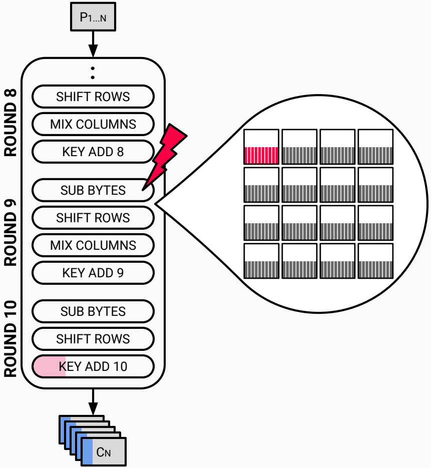
Statistical Fault Attacks on AES in Pictures



Statistical Fault Attacks on AES in Pictures



Statistical Fault Attacks on AES in Pictures



Countermeasures

Categories

- Sensors to detect tampering

Categories

- Sensors to detect tampering
- Protocol/Mode level
  - Limited key usage, no message twice ...

Categories

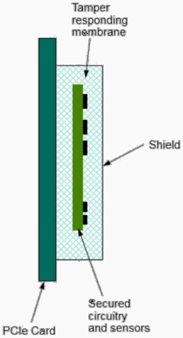
- Sensors to detect tampering
- Protocol/Mode level
  - Limited key usage, no message twice . . .
- Algorithmic countermeasures
  - (Often) no hardware support needed
  - Added redundancy to detect/correct errors
  - Hiding (shuffling, random delays,. . .) to hinder precise fault injection, (masking)

Sensor-based Countermeasures

- Sensors to detect anomalies
    - Active meshes: Fine wire mesh across IC, disruption is detected
    - Power surge sensors
    - Temperature sensors
    - Light sensors

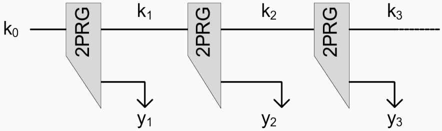
Sensor-based Countermeasures

- Sensors to detect anomalies
  - Active meshes: Fine wire mesh across IC, disruption is detected
  - Power surge sensors
  - Temperature sensors
  - Light sensors
- Example: IBM4767 Hardware Security Module
  - Battery-backed monitoring, meshes, light sensors, temperature sensors, etc.
  - Immediate deletion of keying material on tamper detection



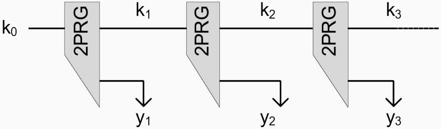
Protocol Level Countermeasures

- Precondition for differential fault attacks
  - Encrypt same message twice with same key, get faulty output
  - Break condition!



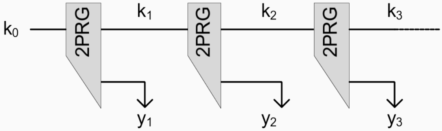
Protocol Level Countermeasures

- Precondition for differential fault attacks
  - Encrypt same message twice with same key, get faulty output
  - Break condition!
- Frequent key update
  - Use key for only 1 encryption, then update



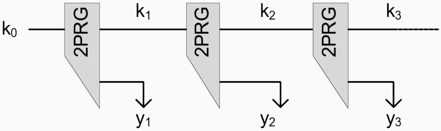
Protocol Level Countermeasures

- Precondition for differential fault attacks
  - Encrypt same message twice with same key, get faulty output
  - Break condition!
- Frequent key update
  - Use key for only 1 encryption, then update
- Protocol that doesn't allow encryption of same message
  - "Proper" modes needs randomization (nonce) anyway
  - Problem: Decryption!



Protocol Level Countermeasures

- Precondition for differential fault attacks
  - Encrypt same message twice with same key, get faulty output
  - Break condition!
- Frequent key update
  - Use key for only 1 encryption, then update
- Protocol that doesn't allow encryption of same message
  - "Proper" modes needs randomization (nonce) anyway
  - Problem: Decryption!
- Authenticated encryption
  - Tag verifies integrity of ciphertext
  - Fault in decryption likely invalidates tag
    → No faulty output is released



Algorithmic Countermeasure: Redundancy

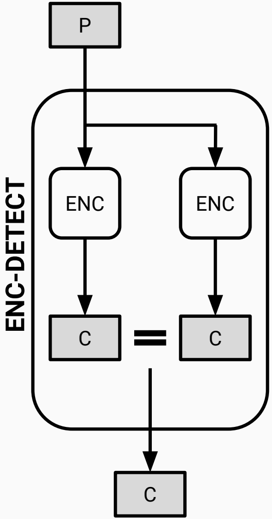
- Old problem in communication: Noisy Channels
  - Receiver wants to detect transmission errors and correct them
  - Now: “Noise” source is attacker instead of channel

Algorithmic Countermeasure: Redundancy

- Old problem in communication: Noisy Channels
  - Receiver wants to detect transmission errors and correct them
  - Now: “Noise” source is attacker instead of channel
- Solution: Redundancy
  - Transmit redundant representation of data
    (more bits than actually needed)
  - Use redundant information for error detection/correction

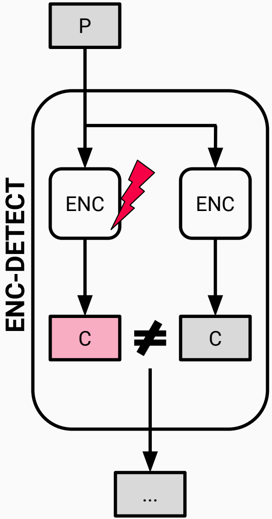
Fault Countermeasures - Detection

- Use redundancy to detect faults



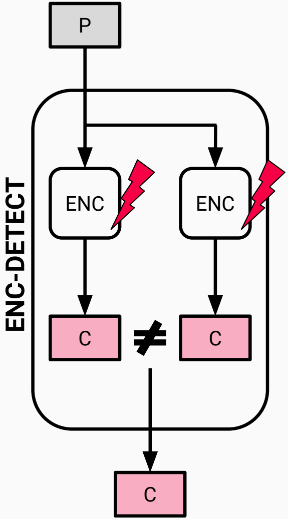
Fault Countermeasures - Detection

- Use redundancy to detect faults
- Fault detected → No ciphertext



Fault Countermeasures - Detection

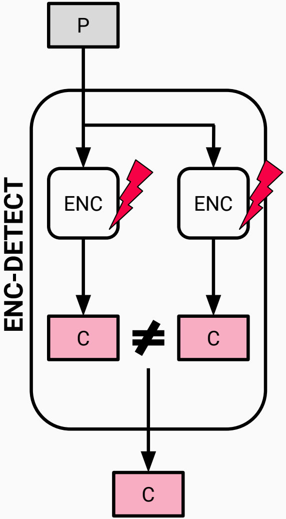
- Use redundancy to detect faults
- Fault detected → No ciphertext
- 2 identical faults necessary for attack



Fault Countermeasures - Detection

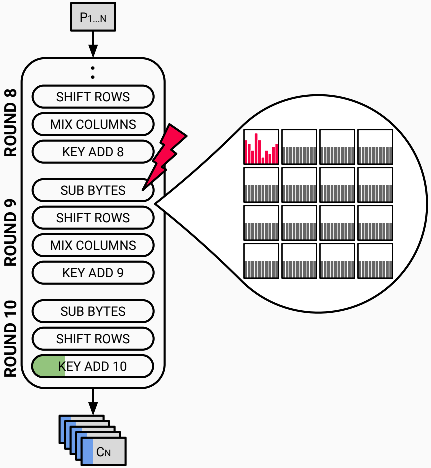
- Use redundancy to detect faults
- Fault detected → No ciphertext
- 2 identical faults necessary for attack

→ More redundancy, Enc-Dec, etc...

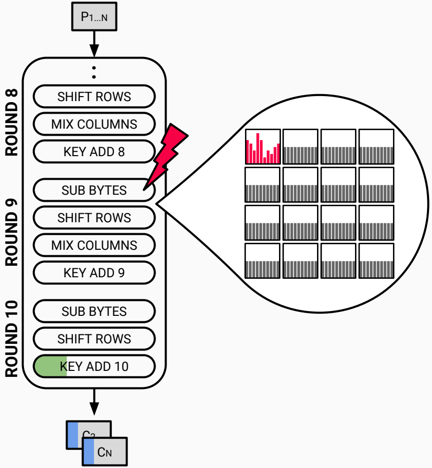


Breaking Countermeasures Again

Statistical Fault Attacks on AES in Pictures



Statistical (Ineffective) Fault Attacks on AES in Pictures

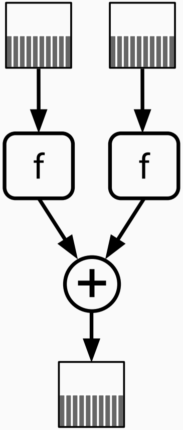


*only correct computations are considered

What about masked redundant implementations?

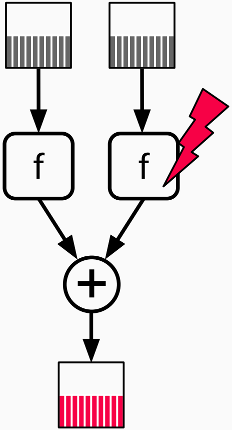
Statistical (Ineffective) Fault Attacks on Masked Redundand AES

- Faulting single shares in linear functions does not work...



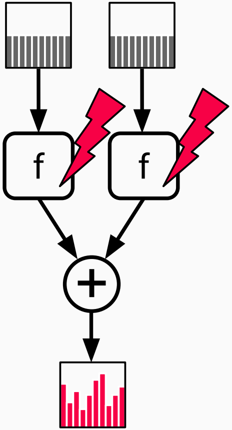
Statistical (Ineffective) Fault Attacks on Masked Redundand AES

- Faulting single shares in linear functions does not work...



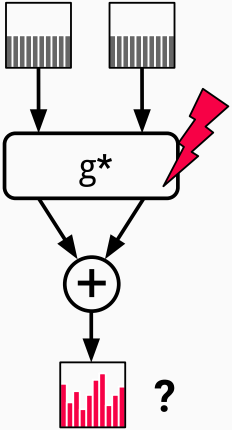
Statistical (Ineffective) Fault Attacks on Masked Redundand AES

- Faulting single shares in linear functions does not work...
- Faulting all shares would work but is difficult...

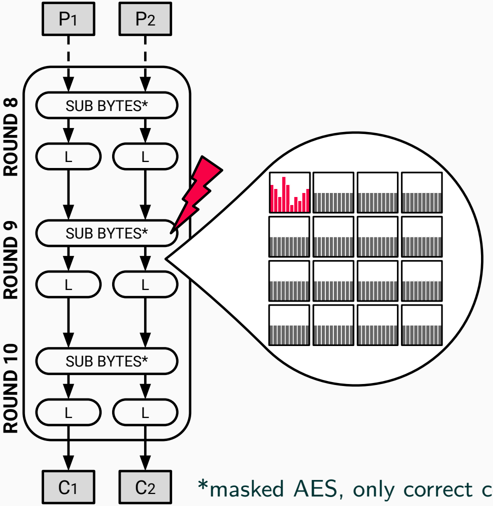


Statistical (Ineffective) Fault Attacks on Masked Redundand AES

- Faulting single shares in linear functions does not work...
- Faulting all shares would work but is difficult...
- Can faulting single shares in non-linear functions lead to a bias in the unshared value?

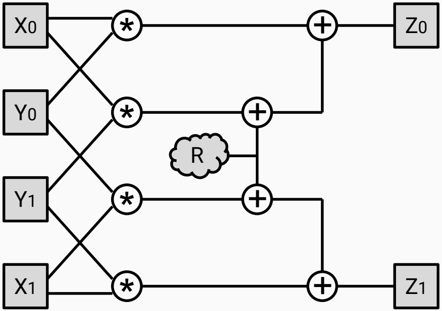


Statistical (Ineffective) Fault Attacks on Redundand Masked AES

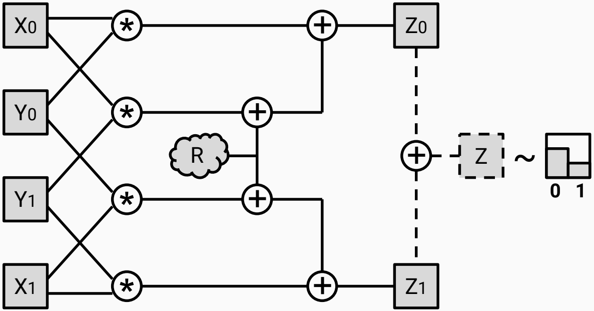


*masked AES, only correct computations are considered

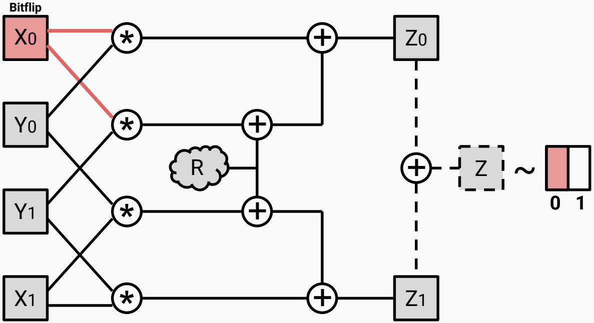
Faults on Masked DOM And Gate



Faults on Masked DOM And Gate

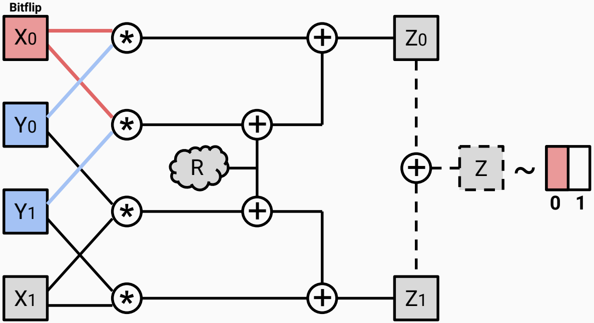


Faults on Masked DOM And Gate



*only correct computations are considered

Faults on Masked DOM And Gate

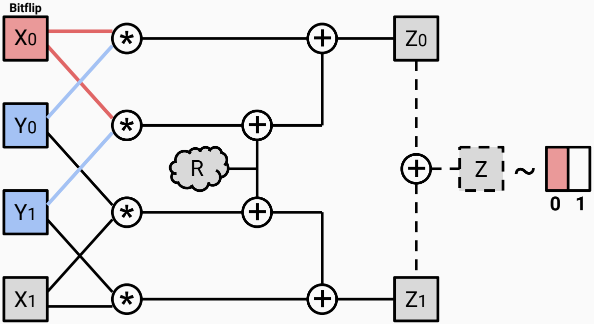


*only correct computations are considered

Faults on Masked AND

Also works with:

- Other types of faults
- Higher-order masking
- Threshold Implementations



*only correct computations are considered

Conclusions

- Fault attacks can be quite tricky to deal with. . .

Conclusions

- Fault attacks can be quite tricky to deal with...
- Can still be mitigated with:
    - Carefully crafted cipher implementations + redundancy (assuming an attacker injects only up to x faults)
    - Cryptographic modes/protocols that limit key usage

Conclusions

- Fault attacks can be quite tricky to deal with. . .
- Can still be mitigated with:
  - Carefully crafted cipher implementations + redundancy (assuming an attacker injects only up to x faults)
  - Cryptographic modes/protocols that limit key usage
- In practice one also often also relies on:
  - Hiding to decrease attack performance
  - Sensor-based countermeasures if available

Remarks

- This is the last actual lecture of SCS
- Monday 24th: EX2 Deadline

Thank you!

Questions:

rishub.nagpal@iaik.tugraz.at

Discord

Fault Attacks

Side-Channel Security

Rishub Nagpal

June 13, 2024

IAIK – Graz University of Technology