Effective Attacks from Ineffective Faults

Maria Eichlseder

Includes results of joint works with Joan Daemen, Christoph Dobraunig, Hannes Groß, Thomas Korak, Stefan Mangard, Florian Mendel, Robert Primas

ISCwsISC, 22 February 2021

SCIENCE
PASSION
TECHNOLOGY

Outline

Introduction to Fault Attacks

- Flipping Bits in Symmetric Crypto
- A Detour to Differential Cryptanalysis

Countermeasures

- Error Detection & Infection
- Fault Attack Variants
- Side-Channel Countermeasures

Statistical Ineffective Fault Attacks

- Why & how SIFA works
- SIFA against masked, redundant implementations

Defending against SIFA

- Criterion for SIFA resistance
- A combined countermeasure

Introduction to Fault Attacks

Causing Faulty Computations

Extreme environmental conditions or targeted manipulations can cause errors in a processor's operation due to physical corruption. Examples:

- Very high temperature
- Unsupported supply voltage or current, voltage glitches
- Overclocking, clock glitches
- Excessive memory accesses
- Strong electric or magnetic fields
- Ionizing radiation
- Laser

Possible Fault Effects

Fault effects in electronic devices have been studied at least since the 1950s, for example for radiation from nuclear testing:

- Long-term effects, e.g., cumulative effect of "Total Ionization Dose (TID)"
- Sudden effects, e.g., charged particle hits the circuit: "Single-Event Effects (SEE)"
  - Causing permanent damage (hard error)
    e.g., shorts between ground and power: "Single-Event Latch-ups (SEL)"
  - Causing temporary damage (soft error)
    e.g., transient pulse flips a bit in memory cell: "Single-Event Upsets (SEU)"

  Some possible effects in processors:
  - Flip a data bit
  - Reset a data bit to 0
  - Skip an instruction

Possible Fault Effects

Fault effects in electronic devices have been studied at least since the 1950s, for example for radiation from nuclear testing:

- ⌛ Long-term effects, e.g., cumulative effect of "Total Ionization Dose (TID)"
- ⚡ Sudden effects, e.g., charged particle hits the circuit: "Single-Event Effects (SEE)"
  - Causing permanent damage (hard error)
    e.g., shorts between ground and power: "Single-Event Latch-ups (SEL)"
  - Causing temporary damage (soft error)
    e.g., transient pulse flips a bit in memory cell: "Single-Event Upsets (SEU)"

Some possible effects in processors:

- Flip a data bit
- Reset a data bit to 0
- Skip an instruction

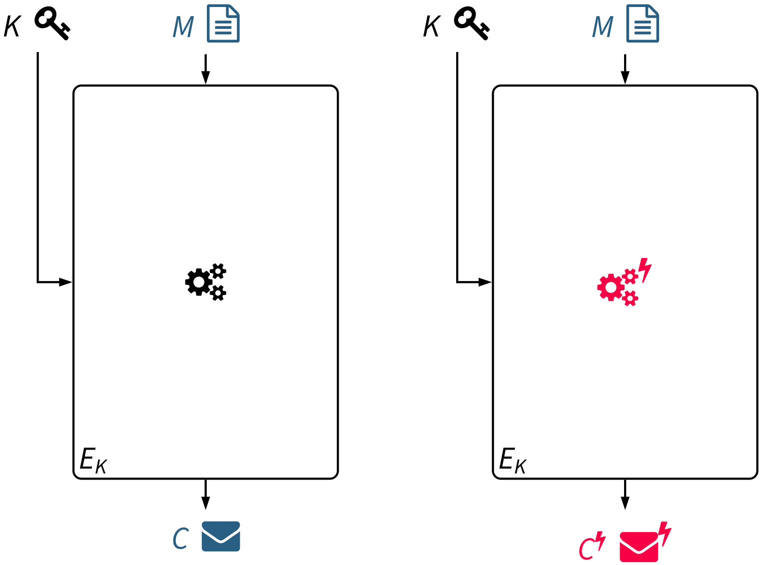
Possible Fault Effects

Fault effects in electronic devices have been studied at least since the 1950s, for example for radiation from nuclear testing:

- Long-term effects, e.g., cumulative effect of "Total Ionization Dose (TID)"
- Sudden effects, e.g., charged particle hits the circuit: "Single-Event Effects (SEE)"
  - Causing permanent damage (hard error)
    e.g., shorts between ground and power: "Single-Event Latch-ups (SEL)"
  - Causing temporary damage (soft error)
    e.g., transient pulse flips a bit in memory cell: "Single-Event Upsets (SEU)"

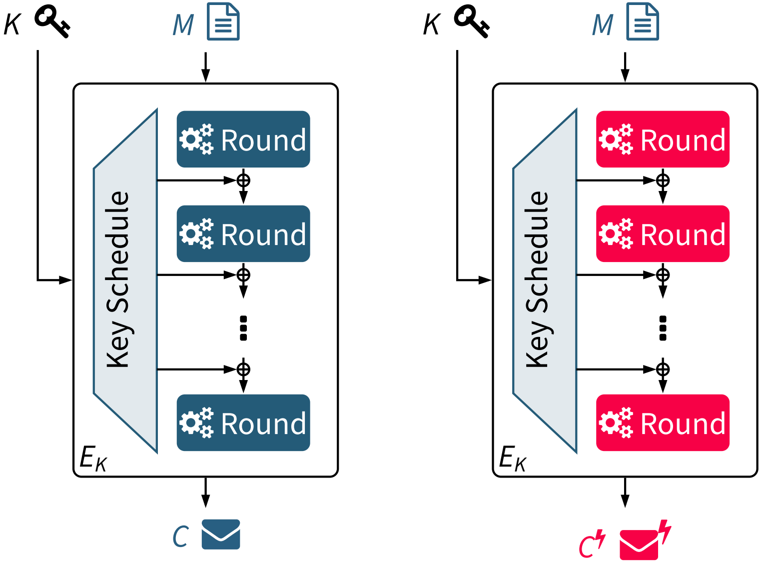
  Some possible effects in processors:
  - Flip a data bit
  - Reset a data bit to 0
  - Skip an instruction

Scenario: Faulting a Block Cipher



- Multiple executions
- Get correct ciphertext C and faulty $C^{\lightning}$

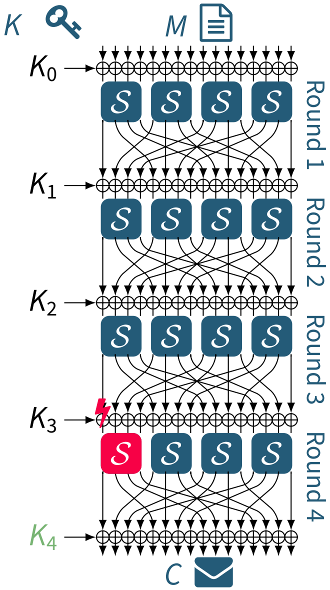
Scenario: Faulting a Block Cipher



- Multiple executions
- Get correct ciphertext C and faulty $C^{ϟ}$

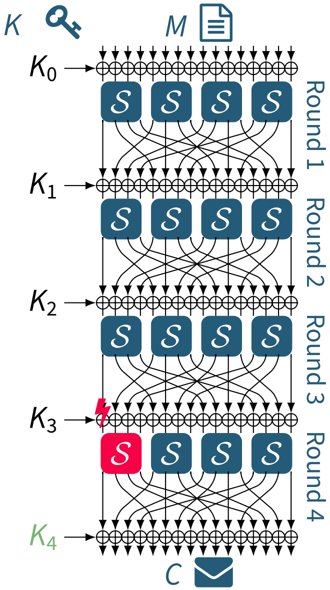
Differential Fault Attacks [BS97]

1. Obtain correct C and faulty $C^{\lightning}$
2. Compute the difference $\Delta C = C \oplus C^{\lightning}$ and derive the output difference of S-box $\mathcal{S}$
3. For each possible guess of (parts of) K_4:
   - Partially decrypt C, $C^{\lightning}$ and check if the observed difference at the input of $\mathcal{S}$ matches the fault model
   - If not, reject key candidate
4. Repeat to further narrow down the keys



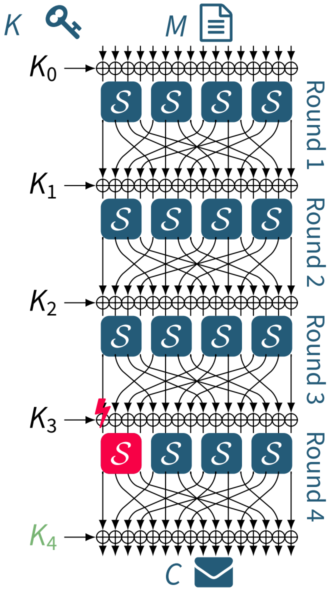
Differential Fault Attacks [BS97]

1. Obtain correct C and faulty $C^{\lightning}$
2. Compute the difference $\Delta C = C \oplus C^{\lightning}$ and derive the output difference of S-box $\mathcal{S}$
3. For each possible guess of (parts of) K_4:
   - Partially decrypt C, $C^{\lightning}$ and check if the observed difference at the input of $\mathcal{S}$ matches the fault model
   - If not, reject key candidate
4. Repeat to further narrow down the keys



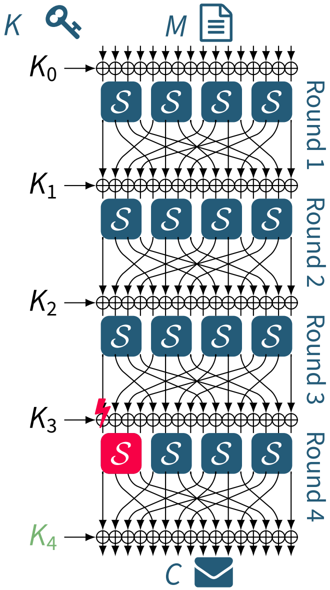
Differential Fault Attacks [BS97]

1. Obtain correct C and faulty $C^{\text{ϟ}}$
2. Compute the difference $\Delta C = C \oplus C^{\text{ϟ}}$ and derive the output difference of S-box $\mathcal{S}$
3. For each possible guess of (parts of) K_4:
   - Partially decrypt C, $C^{\text{ϟ}}$ and check if the observed difference at the input of $\mathcal{S}$ matches the fault model
   - If not, reject key candidate
4. Repeat to further narrow down the keys



Differential Fault Attacks [BS97]

1. Obtain correct C and faulty $C^{\lightning}$
2. Compute the difference $\Delta C = C \oplus C^{\lightning}$ and derive the output difference of S-box $\mathcal{S}$
3. For each possible guess of (parts of) K_4:
   - Partially decrypt C, $C^{\lightning}$ and check if the observed difference at the input of $\mathcal{S}$ matches the fault model
   - If not, reject key candidate
4. Repeat to further narrow down the keys



A Detour to Differential Cryptanalysis

- One of the two most important cryptanalytic attacks for secret-key crypto
  Biham and Shamir [BS90]
- Chosen-plaintext attack (no cheating with the implementation!)
- Main idea:
  1. Predict effect of plaintext difference $\Delta M = M \oplus M^*$ on ciphertext difference $\Delta C = C \oplus C^*$ without knowing K
  2. Use prediction as distinguisher to recover the key

Differential Properties of S-boxes

$\Delta \text{in} = 8 \quad \rightarrow \quad \Delta \text{out} = ?$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	2	0	4	3	9	5	6	7	1	d	e	f	a	8	c	b

Differential Properties of S-boxes

$\Delta\text{in} = 8 \quad \rightarrow \quad \Delta\text{out} = ?$

$\Delta\texttt{in} = 8$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	2	0	4	3	9	5	6	7	1	d	e	f	a	8	c	b

$\Delta\texttt{out} = 3$

Differential Properties of S-boxes

$\Delta\text{in} = 8 \quad \rightarrow \quad \Delta\text{out} = ?$

$\Delta\texttt{in} = 8$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	2	0	4	3	9	5	6	7	1	d	e	f	a	8	c	b

$\Delta\texttt{out} = \texttt{d}$

Differential Properties of S-boxes

$\Delta\text{in} = 8 \quad \rightarrow \quad \Delta\text{out} = ?$

$\Delta\texttt{in} = 8$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	2	0	4	3	9	5	6	7	1	d	e	f	a	8	c	b

$\Delta\texttt{out} = \texttt{a}$

Differential Properties of S-boxes

$\Delta\text{in} = 8 \quad \rightarrow \quad \Delta\text{out} \in \{3, \mathtt{a}, \mathtt{c}, \mathtt{d}\}$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	2	0	4	3	9	5	6	7	1	d	e	f	a	8	c	b

- Knowing the value tells us the difference
- Knowing the difference tells us (something about) the value:

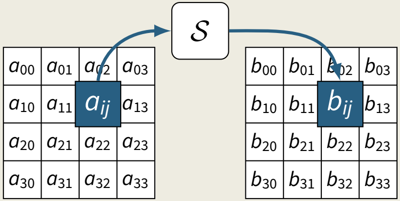
$$\text{solutions}(\Delta\text{in}, \Delta\text{out}) := \{x \;:\; \mathcal{S}(x \oplus \Delta\text{in}) \oplus \mathcal{S}(x) = \Delta\text{out}\}$$

Differential Distribution Table (DDT)

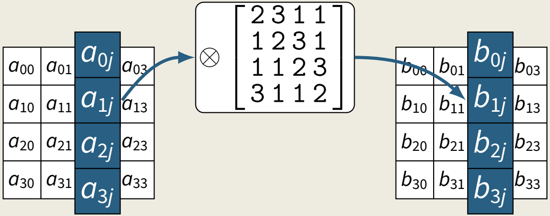
I\O	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	4	4	-	-	-	-	4	-	-	-	-	4	-	-	-
2	-	-	4	4	-	-	4	-	-	-	-	-	-	-	-	4
3	-	4	-	4	4	-	-	-	-	-	-	-	-	-	4	-
4	-	-	4	-	4	4	-	-	-	-	-	4	-	-	-	-
5	-	-	-	4	-	4	-	4	-	4	-	-	-	-	-	-
6	-	-	-	-	4	-	4	4	-	-	-	-	-	4	-	-
7	-	4	-	-	-	4	4	-	-	-	4	-	-	-	-	-
8	-	-	-	4	-	-	-	-	-	-	4	-	4	4	-	-
9	-	4	-	-	-	-	-	-	-	-	-	4	-	4	-	4
a	-	-	-	-	-	4	-	-	-	-	-	-	4	-	4	4
b	-	-	4	-	-	-	-	-	-	4	-	-	-	4	4	-
c	-	-	-	-	-	-	-	-	16	-	-	-	-	-	-	-
d	-	-	-	-	4	-	-	-	-	4	4	-	-	-	-	4
e	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	-

Design of AES [DR02] – Round Function (10 or 12 or 14 Rounds)

1 SubBytes (SB)

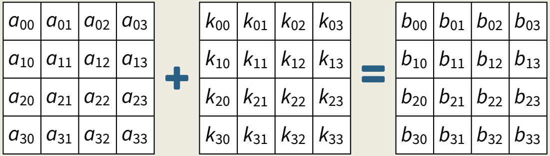


3 MixColumns (MC)

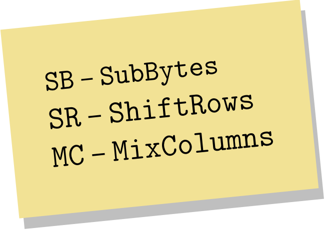


2 ShiftRows (SR)

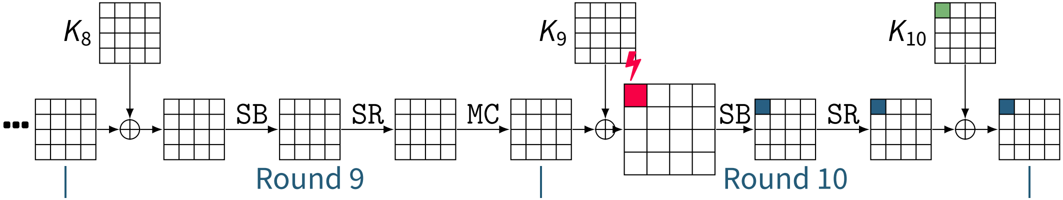
4 AddRoundKey (AK)



AES – Simple DFA



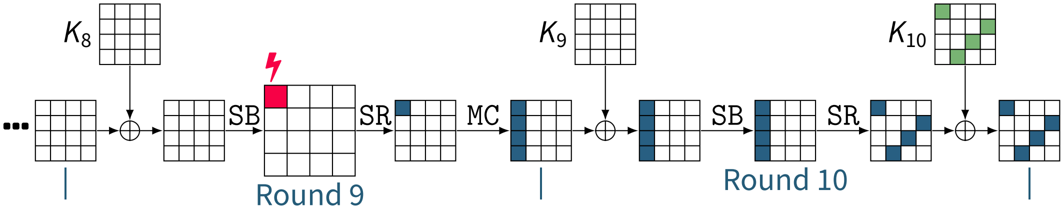
- Assume the attacker can cause precise 1-bit flips in Round 9 of AES, before S-box
- For each of 2^8 key guesses,
  Test if the partial decryption produces the expected 1-bit flip.



AES – Piret and Quisquater's DFA [PQ03]

SB – SubBytes
SR – ShiftRows
MC – MixColumns

- Assume the attacker can cause imprecise 1-byte errors
- For each of 2^{32} key guesses,
  Test if the partial decryption produces the expected 1-byte error.
  (This can be optimized to require only 2 faulty encryptions to recover the full key)



Countermeasures

and Countermeasures against Countermeasures :-)

Types of Countermeasures

Physical level

- Shielding of the circuit so that it’s harder to access
- Sensors that detect tampering

Implementation-level

- Detect or correct errors
- Randomize the execution details

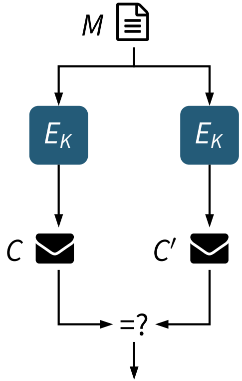
Protocol-level

- Prevent an attacker from collecting useful data by limiting key usage, randomizing inputs, …

Error Detection

- For DFA, the attacker requires the faulty ciphertext $C^{\lightning}$ and the correct ciphertext C for the same plaintext M

- Countermeasure 1: Error Detection
  - Check the correctness of each encryption
  - For example by evaluating it twice
  - Only return result if correct



Error Detection

- For DFA, the attacker requires the faulty ciphertext $C^{\lightning}$ and the correct ciphertext C for the same plaintext M

Countermeasure 2: Authenticated Encryption (AEAD)
AEAD typically prevents DFA by design:

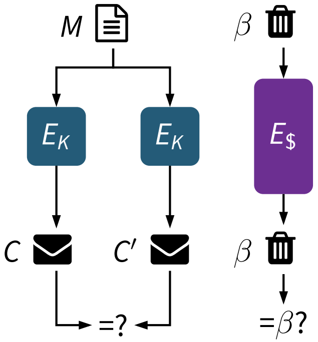
- *E* During AEAD Encryption, a random nonce is used to "randomize" the inputs $M \rightarrow$ cannot get $C, C^{\lightning}$ for the same M
- *D* During AEAD Decryption, results are only returned if the authentication tag was verified correctly, so we usually don't get $C^{\lightning}$

Infection-based Countermeasures

For DFA, the attacker requires the faulty ciphertext $C^{ϟ}$ and the correct ciphertext C for the same plaintext M

Countermeasure 3: Infection

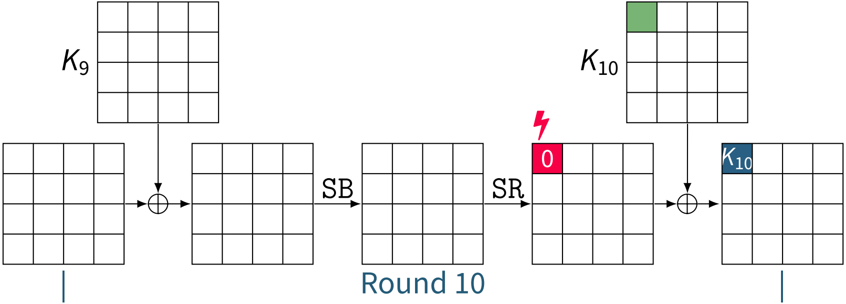
- Do 2 encryptions + many dummy rounds
- If error detected, return dummy garbage
- Can perform checks after every round
- Example for AES: [TBM14]



return if success, else

Ineffective Fault Attacks (IFA) [Cla07] and Friends

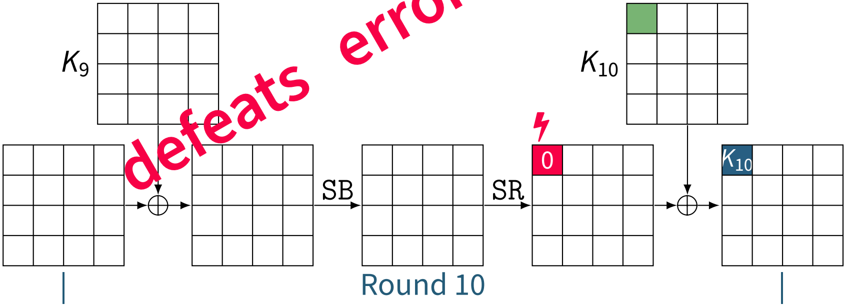
- Observation: In practice, it's often easier to cause biased errors than bitflips
- Example: Stuck-at-0 error sets bit (or byte) to 0
- If the attacker can reliably cause such errors, there are very simple attacks:



Ineffective Fault Attacks (IFA) [Cla07] and Friends

- Observation: In practice, it's often easier to cause biased errors than bitflips
- Example: Stuck-at-0 error sets bit (or byte) to 0
- If the attacker can reliably cause such errors, there are very simple attacks:

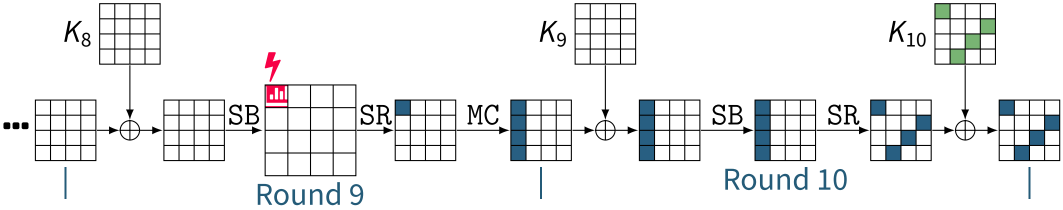
defeats error detection



Statistical Fault Attacks (SFA) [FJLT13]

- Assume the attacker can cause a biased error (e.g., reset to 0 with prob. $\frac{1}{2}$).
- For each of 2^{32} key guesses,
  Test if the partial decryption produces a non-uniform distribution with a metric such as the Squared Euclidean Imbalance (SEI) or Pearson's χ^2:

$$\mathrm{SEI}(\hat{p}) = \sum_{x \in \mathcal{X}} \left| \hat{p}(x) - \frac{1}{\#\mathcal{X}} \right|^2$$



Side-Channel Countermeasures

IFA allows to “peek” at intermediate values, similar to side-channel attacks.

Many side-channel countermeasures help against IFA and friends:

- Hiding: Randomize the order of instructions, insert dummy instructions, etc., to make it harder for the attacker to hit the right bit
- Masking: Replace each data bit x by $d + 1$ random bits $x_0, x_1, \ldots, x_d$ with

$$x = x_0 \oplus x_1 \oplus \ldots \oplus x_d$$

  Then learning up to d bits x_i is useless for the attacker.

Statistical Ineffective Fault Attacks

Statistical Ineffective Fault Attacks (SIFA) [DEK+18; DEG+18]

So far, we inserted faults right before / after S-boxes.
When the attacker can only place 1 fault, error detection and/or masking prevent these attacks.

SIFA idea 1: Use only faulty encryptions where **no fault was detected**:
This condition may lead to a **bias** in some intermediate variables!

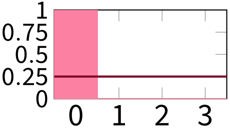
SIFA idea 2: Place **fault inside** the S-box circuit,
but **measure before/after** S-box with SFA methods!

This approach can attack implementations with masking and error detection.
It may, however, require more data (1000s of messages).

SIFA Idea 1: Ineffective Faults & Fault Distribution Tables

How are values distributed if we consider only ineffective faults $X^{\lightning} = X$?

X \ $X^{\lightning}$	00	01	10	11
00	1	0	0	0
01	1	0	0	0
10	1	0	0	0
11	1	0	0	0



(a) Stuck-at-0

X \ $X^{\lightning}$	00	01	10	11
00	1	0	0	0
01	$\frac{1}{2}$	$\frac{1}{2}$	0	0
10	$\frac{1}{2}$	0	$\frac{1}{2}$	0
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

(b) Random-AND

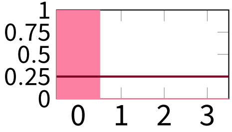
X \ $X^{\lightning}$	00	01	10	11
00	0	0	0	1
01	0	0	1	0
10	0	1	0	0
11	1	0	0	0

(c) Bit-flip

SIFA Idea 1: Ineffective Faults & Fault Distribution Tables

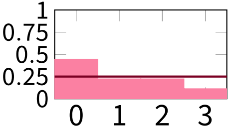
How are values distributed if we consider only ineffective faults $X^{\lightning} = X$?

X \ $X^{\lightning}$	00	01	10	11
00	1	0	0	0
01	1	0	0	0
10	1	0	0	0
11	1	0	0	0



(a) Stuck-at-0

X \ $X^{\lightning}$	00	01	10	11
00	1	0	0	0
01	$\frac{1}{2}$	$\frac{1}{2}$	0	0
10	$\frac{1}{2}$	0	$\frac{1}{2}$	0
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$



(b) Random-AND

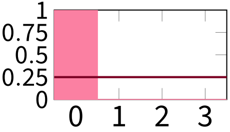
X \ $X^{\lightning}$	00	01	10	11
00	0	0	0	1
01	0	0	1	0
10	0	1	0	0
11	1	0	0	0

(c) Bit-flip

SIFA Idea 1: Ineffective Faults & Fault Distribution Tables

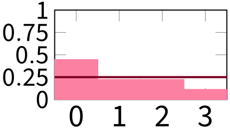
How are values distributed if we consider only ineffective faults $X^{\lightning} = X$?

x \ $x^{\lightning}$	00	01	10	11
00	1	0	0	0
01	1	0	0	0
10	1	0	0	0
11	1	0	0	0



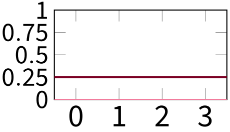
(a) Stuck-at-0

x \ $x^{\lightning}$	00	01	10	11
00	1	0	0	0
01	$\frac{1}{2}$	$\frac{1}{2}$	0	0
10	$\frac{1}{2}$	0	$\frac{1}{2}$	0
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$



(b) Random-AND

x \ $x^{\lightning}$	00	01	10	11
00	0	0	0	1
01	0	0	1	0
10	0	1	0	0
11	1	0	0	0



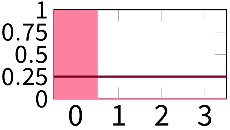
(c) Bit-flip

SIFA Idea 1: Ineffective Faults & Fault Distribution Tables

How are values distributed if we consider only ineffective faults $X^{\lightning} = X$?

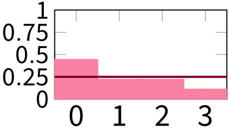
non-uniform distribution

x \ $x^{\lightning}$	00	01	10	11
00	1	0	0	0
01	1	0	0	0
10	1	0	0	0
11	1	0	0	0



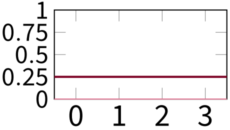
(a) Stuck-at-0

x \ $x^{\lightning}$	00	01	10	11
00	1	0	0	0
01	$\frac{1}{2}$	$\frac{1}{2}$	0	0
10	$\frac{1}{2}$	0	$\frac{1}{2}$	0
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$



(b) Random-AND

x \ $x^{\lightning}$	00	01	10	11
00	0	0	0	1
01	0	0	1	0
10	0	1	0	0
11	1	0	0	0



(c) Bit-flip

SIFA Idea 1: Ineffective Faults & Fault Distribution Tables

1. Inject fault with non-uniform distribution $p_{\text{eq}}(x^{\lightning}) = \mathbb{P}[X^{\lightning} = x^{\lightning} \mid X^{\lightning} = X]$
2. Keep only samples where no error was detected (ineffective fault, like IFA)
   - Fault Ineffectivity Rate $\pi_{\text{eq}} = \mathbb{P}[X^{\lightning} = X]$ is the ratio of these samples
3. Guess part of key and compute backwards as before
4. Statistically test distribution $p_{\text{eq}}(x^{\lightning})$ like SFA: is it non-uniform?
   - CHI (Pearson's χ^2) or SEI (Squared Euclidean Imbalance)
   - LLR (log-likelihood ratio) if ineffective distribution $p_{\text{eq}}(\cdot)$ is known
5. If it looks uniform, reject key candidate; if non-uniform, keep it

This also works if the fault induction method is noisy
(only works sometimes, with probability σ)

SIFA Idea 1: Ineffective Faults & Fault Distribution Tables

1. Inject fault with non-uniform distribution $p_{\text{eq}}(x^{\lightning}) = \mathbb{P}[X^{\lightning} = x^{\lightning} \mid X^{\lightning} = X]$
2. Keep only samples where no error was detected (ineffective fault, like IFA)
   - Fault Ineffectivity Rate $\pi_{\text{eq}} = \mathbb{P}[X^{\lightning} = X]$ is the ratio of these samples
3. Guess part of key and compute backwards as before
4. Statistically test distribution $p_{\text{eq}}(x^{\lightning})$ like SFA: is it non-uniform?
   - CHI (Pearson's χ^2) or SEI (Squared Euclidean Imbalance)
   - LLR (log-likelihood ratio) if ineffective distribution $p_{\text{eq}}(\cdot)$ is known
5. If it looks uniform, reject key candidate; if non-uniform, keep it

This also works if the fault induction method is noisy
(only works sometimes, with probability σ)

SIFA Idea 1: Ineffective Faults & Fault Distribution Tables

1. Inject fault with non-uniform distribution $p_{\text{eq}}(x^{ϟ}) = \mathbb{P}[X^{ϟ} = x^{ϟ} \mid X^{ϟ} = X]$
2. Keep only samples where no error was detected (ineffective fault, like IFA)
   - Fault Ineffectivity Rate $\pi_{\text{eq}} = \mathbb{P}[X^{ϟ} = X]$ is the ratio of these samples
3. Guess part of key and compute backwards as before
4. Statistically test distribution $p_{\text{eq}}(x^{ϟ})$ like SFA: is it non-uniform?
   - CHI (Pearson's χ^2) or SEI (Squared Euclidean Imbalance)
   - LLR (log-likelihood ratio) if ineffective distribution $p_{\text{eq}}(\cdot)$ is known
5. If it looks uniform, reject key candidate; if non-uniform, keep it

This also works if the fault induction method is noisy
(only works sometimes, with probability σ)

Example: Bytewise Random-AND and Infection Countermeasure

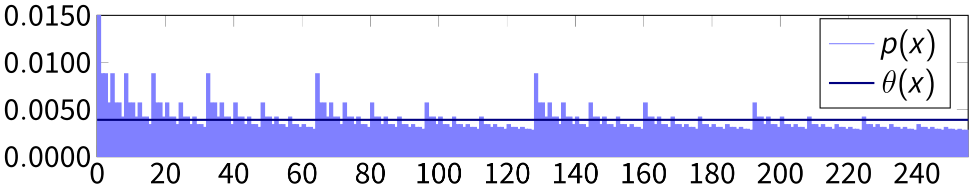
- Fault model: Bytewise fault that flips each 1 to 0 with probability $\frac{1}{2}$
  - Fault ineffectivity rate: $\pi_{eq} = (3/4)^8 \approx 10\,\%$
- Implementation: AES + infection countermeasure, target round 40 of 22+22=44
  - Hit a suitable round with prob. $\sigma \approx 0.315$ among ineffective samples.
  - Distribution $p_{eq}(x)$ for correct key and uniform distribution θ:

$$p_{eq}(x) = \sigma \cdot 2^{8-\mathrm{hw}(x)}/3^8 + (1-\sigma) \cdot 2^{-8}.$$

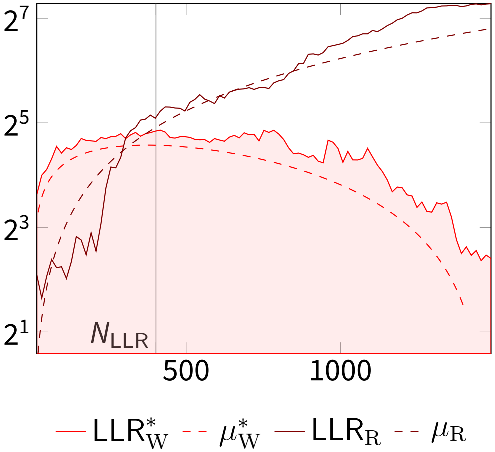
Example: Bytewise Random-AND and Infection Countermeasure

- **Fault model**: Bytewise fault that flips each 1 to 0 with probability $\frac{1}{2}$
  - Fault ineffectivity rate: $\pi_{\text{eq}} = (3/4)^8 \approx 10\,\%$
- **Implementation**: AES + infection countermeasure, target round 40 of 22+22=44
  - Hit a suitable round with prob. $\sigma \approx 0.315$ among ineffective samples.
  - Distribution $p_{\text{eq}}(x)$ for correct key and uniform distribution θ:

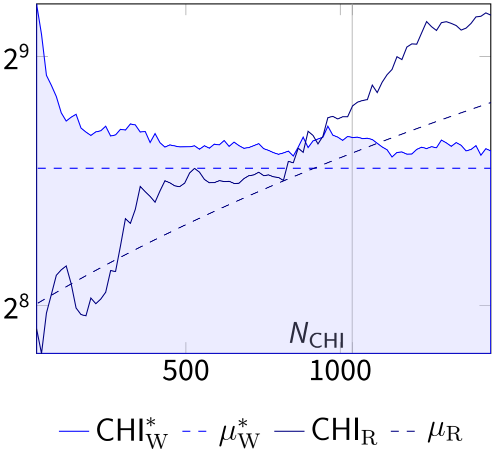
$$p_{\text{eq}}(x) = \sigma \cdot 2^{8-\text{hw}(x)}/3^8 + (1-\sigma) \cdot 2^{-8}.$$



Example: Bytewise Random-AND and Infection Countermeasure



(a) LLR($\hat{p}$) statistic



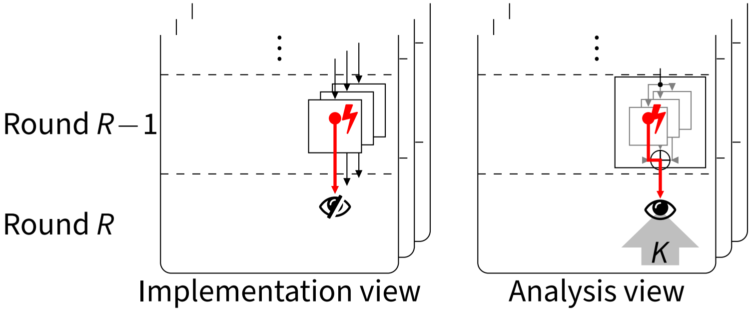
(b) CHI($\hat{p}$) statistic (similar to SEI)

SIFA Idea 2: Faulting Inside an S-box

- So far, we placed the fault before the S-box and tested at the same position
- We can also place the fault inside the S-box and test at the input or output

- Can turn bitflip faults into nice non-uniform ineffective distributions
- Can work even for implementations protected with masking

SIFA on Masked Implementations with Detection Countermeasures



SIFA Example: Inside a Masked S-box Circuit

- Example S-box: A smaller version of SHA-3's S-box (χ)
- 3-bit input a, b, c, masked as
  - $a = a_0 \oplus a_1$
  - $b = b_0 \oplus b_1$
  - $c = c_0 \oplus c_1$
- 3-bit output r, s, t, masked as
  - $r = r_0 \oplus r_1$
  - $s = s_0 \oplus s_1$
  - $t = t_0 \oplus t_1$
- Implemented as circuit of instructions / gates XOR $\oplus$, AND $\odot$, NOT $\ominus$

SIFA Example: Inside a Masked S-box Circuit

Input: $\{a_0, a_1, b_0, b_1, c_0, c_1\}$

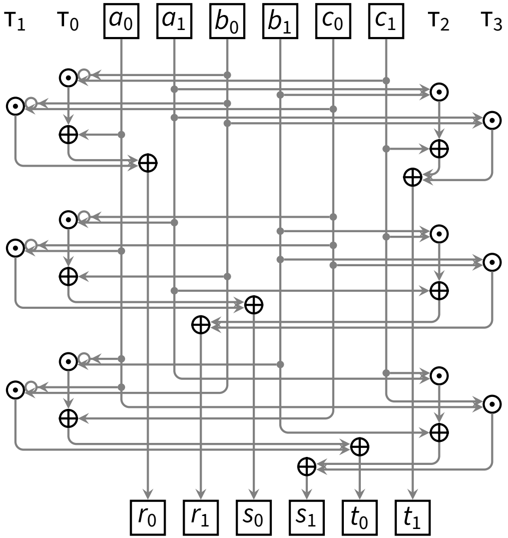
$$
\begin{array}{lllll}
\mathrm{T}_0 \leftarrow \overline{b_0} \odot c_1 & ; & \mathrm{T}_2 \leftarrow a_1 \odot b_1 \\
\mathrm{T}_1 \leftarrow \overline{b_0} \odot c_0 & ; & \mathrm{T}_3 \leftarrow a_1 \odot b_0 \\
\mathrm{T}_0 \leftarrow \mathrm{T}_0 \oplus a_0 & ; & \mathrm{T}_2 \leftarrow \mathrm{T}_2 \oplus c_1 \\
r_0 \leftarrow \mathrm{T}_0 \oplus \mathrm{T}_1 & ; & t_1 \leftarrow \mathrm{T}_2 \oplus \mathrm{T}_3
\end{array}
$$

$$
\begin{array}{lllll}
\mathrm{T}_0 \leftarrow \overline{c_0} \odot a_1 & ; & \mathrm{T}_2 \leftarrow b_1 \odot c_1 \\
\mathrm{T}_1 \leftarrow \overline{c_0} \odot a_0 & ; & \mathrm{T}_3 \leftarrow b_1 \odot c_0 \\
\mathrm{T}_0 \leftarrow \mathrm{T}_0 \oplus b_0 & ; & \mathrm{T}_2 \leftarrow \mathrm{T}_2 \oplus a_1 \\
s_0 \leftarrow \mathrm{T}_0 \oplus \mathrm{T}_1 & ; & r_1 \leftarrow \mathrm{T}_2 \oplus \mathrm{T}_3
\end{array}
$$

ϟ a_0

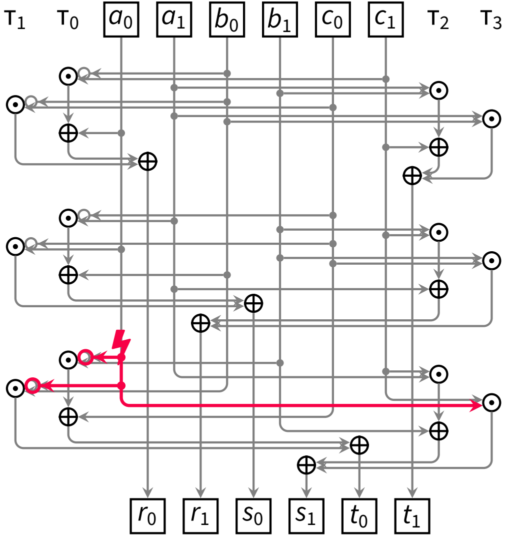
$$
\begin{array}{lllll}
\mathrm{T}_0 \leftarrow \overline{a_0} \odot b_1 & ; & \mathrm{T}_2 \leftarrow c_1 \odot a_1 \\
\mathrm{T}_1 \leftarrow \overline{a_0} \odot b_0 & ; & \mathrm{T}_3 \leftarrow c_1 \odot a_0 \\
\mathrm{T}_0 \leftarrow \mathrm{T}_0 \oplus c_0 & ; & \mathrm{T}_2 \leftarrow \mathrm{T}_2 \oplus b_1 \\
t_0 \leftarrow \mathrm{T}_0 \oplus \mathrm{T}_1 & ; & s_1 \leftarrow \mathrm{T}_2 \oplus \mathrm{T}_3
\end{array}
$$

Output: $\{r_0, r_1, s_0, s_1, t_0, t_1\}$



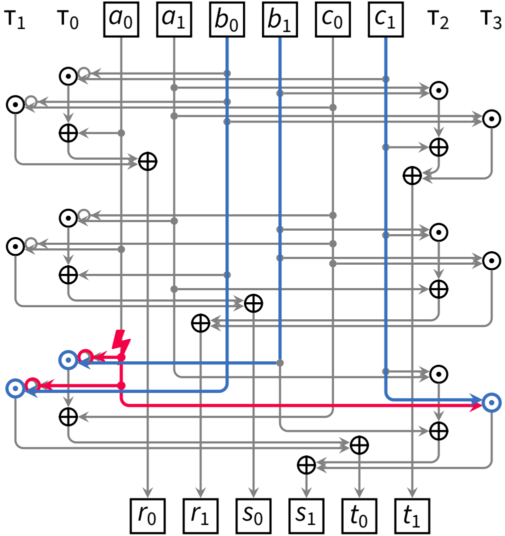
SIFA Example: Inside a Masked S-box Circuit

- Cause a bitflip fault in ϟa_0 at the indicated moment
- The faulty value goes into 3 ⊙s
- Correctness of the ⊙-output depends on the other input
  - if the other input is 0, the ⊙-output is correct
  - if the other input is 1, the ⊙-output is faulty

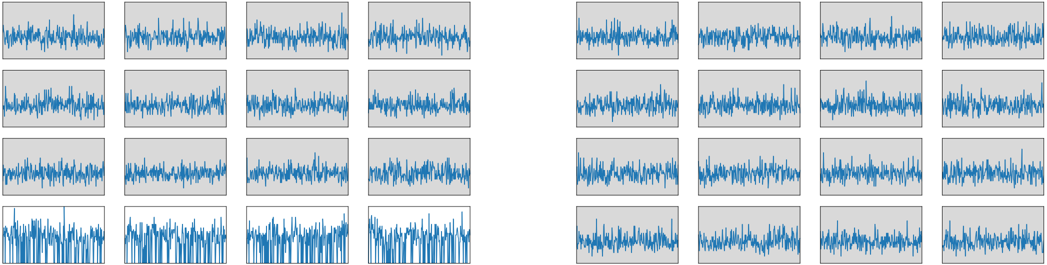


SIFA Example: Inside a Masked S-box Circuit

- The S-box output is correct if $\odot$ with c_1 is correct and
  - both $\odot$s with b_0, b_1 are correct: $b_0 = b_1 = 0$, or
  - both $\odot$s with b_0, b_1 are faulty: $b_0 = b_1 = 1$
- Either way, $b = b_0 \oplus b_1 = 0$
- If the cipher output is correct, learn $b = 0$ (bias)
- Use as before to recover the key!



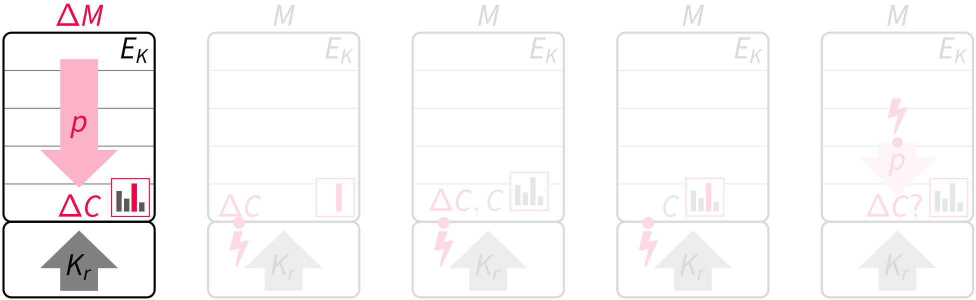
SIFA Example: Application to AES



(a) Correct key guess (b) Wrong key guess

Figure: Results for bitsliced AES implementation on 32-bit platform (ARM Cortex M4) with masking (1st order) and error detection (temporal redundancy). Simulated byte-stuck-at-0 faults. Recovered distribution after S-box in round 9. [DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



Diff. cryptanalysis DC [BS90]

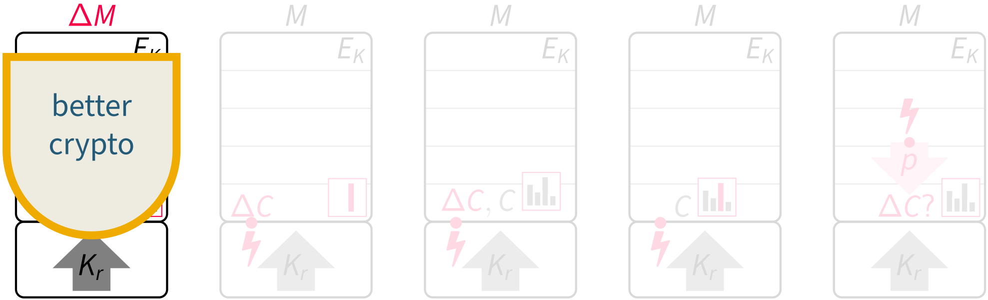
Diff. fault attack DFA [BS97]

Stat. fault attack SFA [FJLT13; DEK+16]

Ineff. fault attack IFA [Cla07]

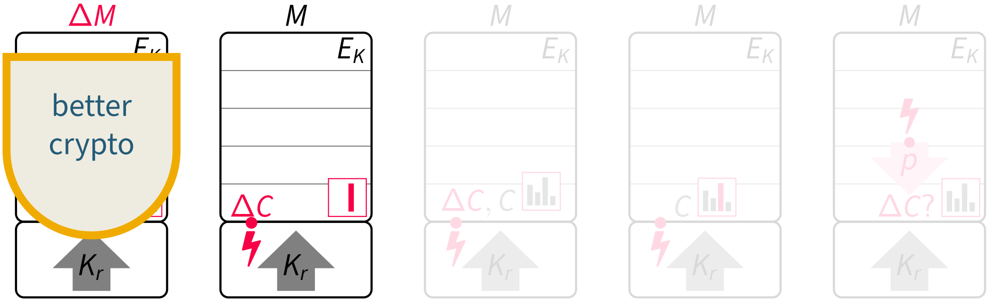
Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



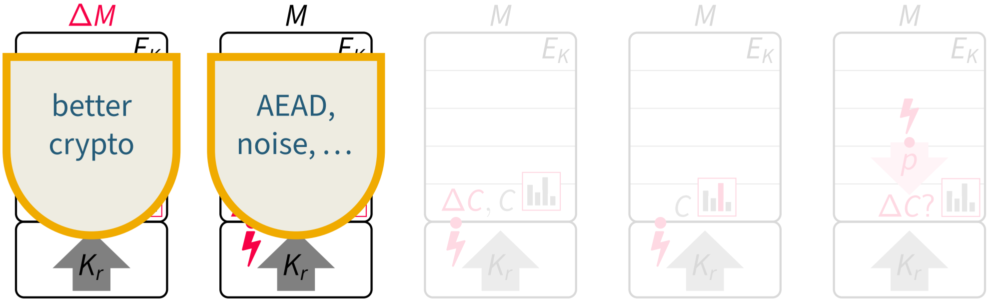
Diff. cryptanalysis DC [BS90]	Diff. fault attack DFA [BS97]	Stat. fault attack SFA [FJLT13; DEK+16]	Ineff. fault attack IFA [Cla07]	Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



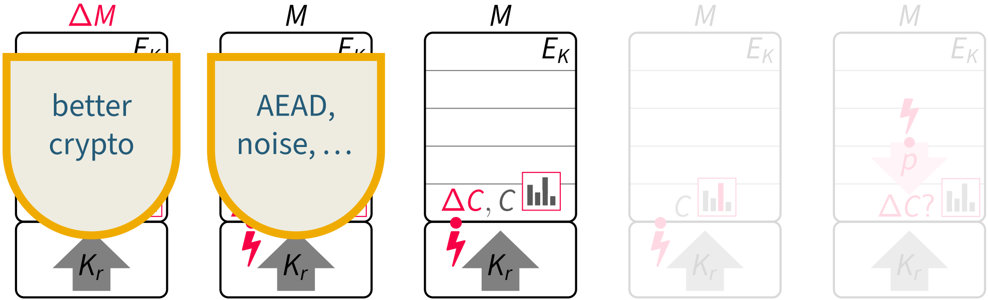
Diff. cryptanalysis DC [BS90]	Diff. fault attack DFA [BS97]	Stat. fault attack SFA [FJLT13; DEK+16]	Ineff. fault attack IFA [Cla07]	Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



Diff. cryptanalysis DC [BS90]	Diff. fault attack DFA [BS97]	Stat. fault attack SFA [FJLT13; DEK+16]	Ineff. fault attack IFA [Cla07]	Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



Diff. cryptanalysis DC [BS90]

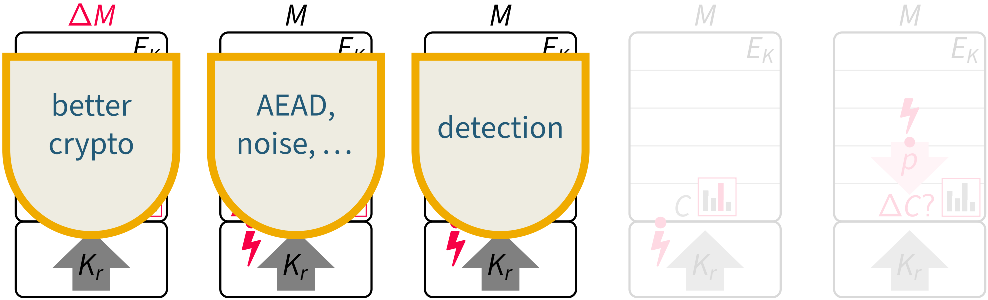
Diff. fault attack DFA [BS97]

Stat. fault attack SFA [FJLT13; DEK+16]

Ineff. fault attack IFA [Cla07]

Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



Diff. cryptanalysis DC [BS90]

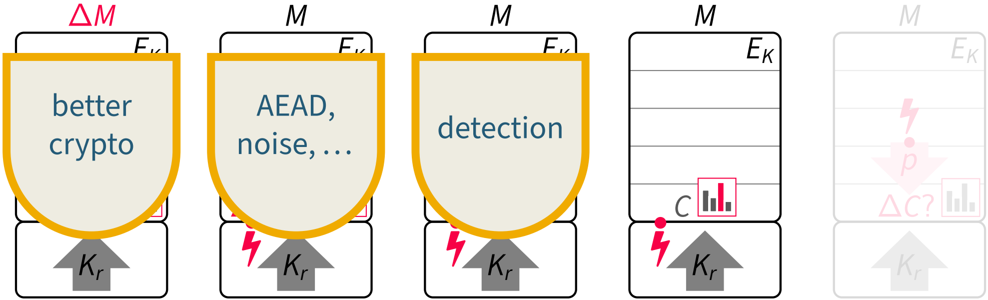
Diff. fault attack DFA [BS97]

Stat. fault attack SFA [FJLT13; DEK+16]

Ineff. fault attack IFA [Cla07]

Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



Diff. cryptanalysis DC [BS90]

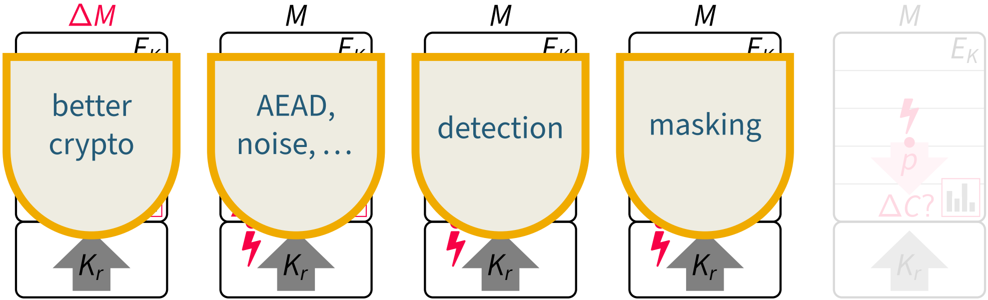
Diff. fault attack DFA [BS97]

Stat. fault attack SFA [FJLT13; DEK+16]

Ineff. fault attack IFA [Cla07]

Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



Diff. cryptanalysis DC [BS90]

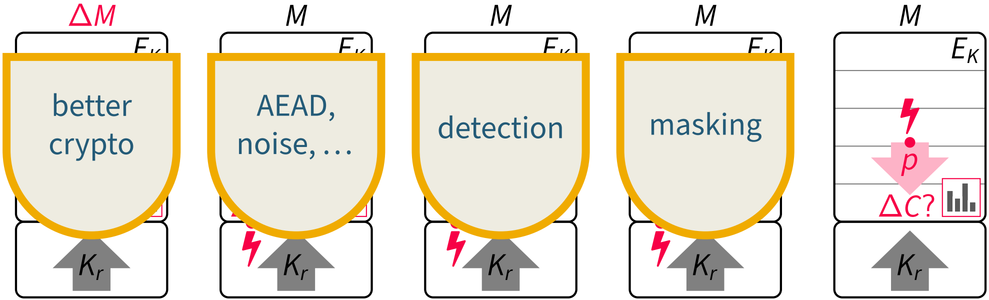
Diff. fault attack DFA [BS97]

Stat. fault attack SFA [FJLT13; DEK+16]

Ineff. fault attack IFA [Cla07]

Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Statistical (Ineffective) Fault Attacks – Summary



Diff. cryptanalysis DC [BS90]	Diff. fault attack DFA [BS97]	Stat. fault attack SFA [FJLT13; DEK+16]	Ineff. fault attack IFA [Cla07]	Statistical Ineffective Fault Attack SIFA [DEK+18; DEG+18]

Defending against SIFA

SIFA Resistance

In a masked implementation, the gates are all incomplete operations: learning all inputs of one gate is not sufficient to learn all shares of one variable.

SIFA on masked implementations works because the fault can

1. propagate to several nonlinear gates and then
2. disappear depending on the other inputs of all these gates.

This way, the effectivity of the fault can depend on all shares of a variable and “reveal” this variable as a non-uniform distribution in the unmasked variables.

An implementation is single-fault SIFA-resistant if each possible single fault

- is either detected by error detection
- or activates (propagates to) at most one nonlinear gate.

SIFA Resistance

In a masked implementation, the gates are all incomplete operations: learning all inputs of one gate is not sufficient to learn all shares of one variable.

SIFA on masked implementations works because the fault can

1. propagate to several nonlinear gates and then
2. disappear depending on the other inputs of all these gates.

This way, the effectivity of the fault can depend on all shares of a variable and "reveal" this variable as a non-uniform distribution in the unmasked variables.

An implementation is single-fault SIFA-resistant if each possible single fault

- is either detected by error detection
- or activates (propagates to) at most one nonlinear gate.

Building SIFA-Resistant Implementations [DDE+20]

Two variants for error detection between 2 redundant computations:

- **Local checks**: Compare relevant intermediate variables during computation
  - One approach: Analyze circuit graph to identify critical variables
  - Easier to develop, but may require many checks

- **Global checks**: Compare only the final unmasked cipher output
  - Need to ensure that all relevant faults propagate to the output
  - One approach: Use only invertible gates like the Toffoli gate
  - More elegant and flexible, but sometimes hard/impossible to develop

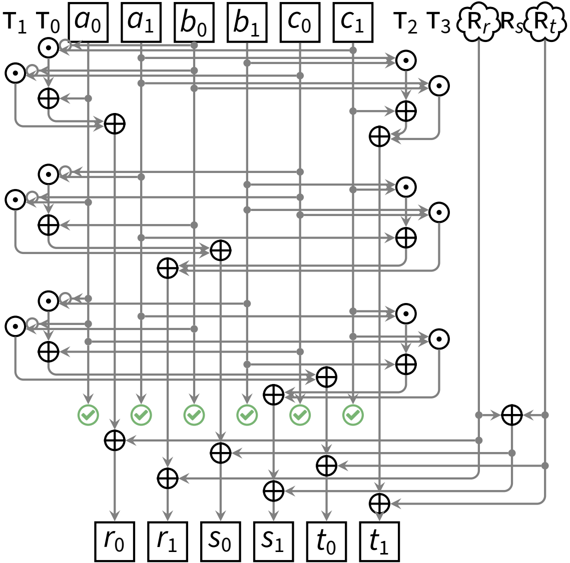
Building SIFA-Resistant Implementations [DDE+20]

Two variants for error detection between 2 redundant computations:

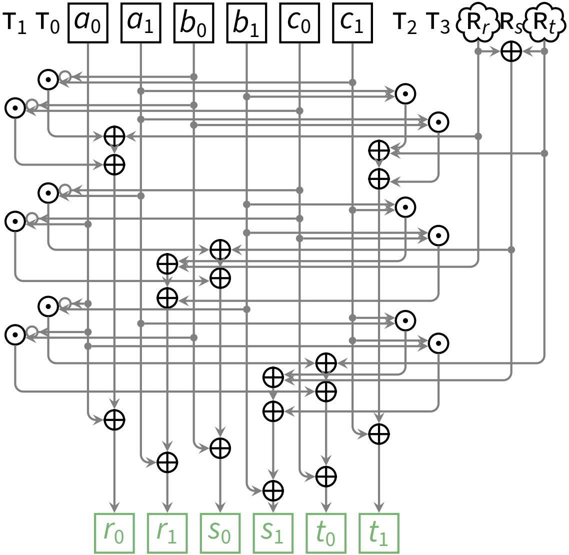
- Local checks: Compare relevant intermediate variables during computation
  - One approach: Analyze circuit graph to identify critical variables
  - Easier to develop, but may require many checks

- Global checks: Compare only the final unmasked cipher output
  - Need to ensure that all relevant faults propagate to the output
  - One approach: Use only invertible gates like the Toffoli gate
  - More elegant and flexible, but sometimes hard/impossible to develop

Example: Single-fault SIFA-resistant χ_3, 2 shares, local checks



Example: Single-fault SIFA-resistant χ_3, 2 shares, global checks



Conclusion

- Statistical Ineffective Fault Attacks are a very powerful type of fault attacks
- Effective against state-of-the-art countermeasures including error detection and side-channel countermeasures (hiding, masking)
- New countermeasures needed
  - Proposal by Daemen et al. [DDE+20]: combine masking & detection with special circuit structure (local checks and/or Toffoli gates)
  - Several other approaches with varying effectivity and efficiency have been published
- With enough effort (money, time, data), attackers may be able to defeat countermeasures – make sure this effort is higher than it’s worth!

Questions

Bibliography I

[BS90] Eli Biham and Adi Shamir. **Differential Cryptanalysis of DES-like Cryptosystems.** Advances in Cryptology – CRYPTO 1990. Vol. 537. LNCS. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1.

[BS97] Eli Biham and Adi Shamir. **Differential Fault Analysis of Secret Key Cryptosystems.** Advances in Cryptology – CRYPTO '97. Vol. 1294. LNCS. Springer, 1997, pp. 513–525. DOI: 10.1007/BFb0052259.

[Cla07] Christophe Clavier. **Secret External Encodings Do Not Prevent Transient Fault Analysis.** Cryptographic Hardware and Embedded Systems – CHES 2007. Vol. 4727. LNCS. Springer, 2007, pp. 181–194. DOI: 10.1007/978-3-540-74735-2_13.

[DDE+20] Joan Daemen, Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Florian Mendel, and Robert Primas. **Protecting against Statistical Ineffective Fault Attacks.** IACR Trans. Cryptogr. Hardw. Embed. Syst. 2020.3 (2020), pp. 508–543. DOI: 10.13154/tches.v2020.i3.508-543.

Bibliography II

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. **Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures.** Advances in Cryptology – ASIACRYPT 2018. Vol. 11273. LNCS. Springer, 2018, pp. 315–342. DOI: 10.1007/978-3-030-03329-3_11.

[DEK+16] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. **Statistical Fault Attacks on Nonce-Based Authenticated Encryption Schemes.** Advances in Cryptology – ASIACRYPT 2016. Vol. 10031. LNCS. Springer, 2016, pp. 369–395. DOI: 10.1007/978-3-662-53887-6_14.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. **SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography.** IACR Transactions on Cryptographic Hardware and Embedded Systems 2018.3 (2018), pp. 547–572. DOI: 10.13154/tches.v2018.i3.547-572.

Bibliography III

[DR02] Joan Daemen and Vincent Rijmen. **The Design of Rijndael: AES – The Advanced Encryption Standard.** Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4.

[FJLT13] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. **Fault Attacks on AES with Faulty Ciphertexts Only.** Fault Diagnosis and Tolerance in Cryptography – FDTC 2013. IEEE Computer Society, 2013, pp. 108–118. DOI: 10.1109/FDTC.2013.18.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. **A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD.** Cryptographic Hardware and Embedded Systems – CHES 2003. Vol. 2779. LNCS. Springer, 2003, pp. 77–88. DOI: 10.1007/978-3-540-45238-6_7.

Bibliography IV

[TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. **Destroying Fault Invariant with Randomization – A Countermeasure for AES Against Differential Fault Attacks.** Cryptographic Hardware and Embedded Systems – CHES 2014. Vol. 8731. LNCS. Springer, 2014, pp. 93–111.