A Comparative Cost/Security Analysis of Fault Attack Countermeasures

T. G. Malkin, F.-X. Standaert, M. Yung

Workshop on Fault Detection and Tolerance in Cryptography, August 2005

Outline

- Introduction
- Error detection codes
- Repetition/duplication
- Conclusion

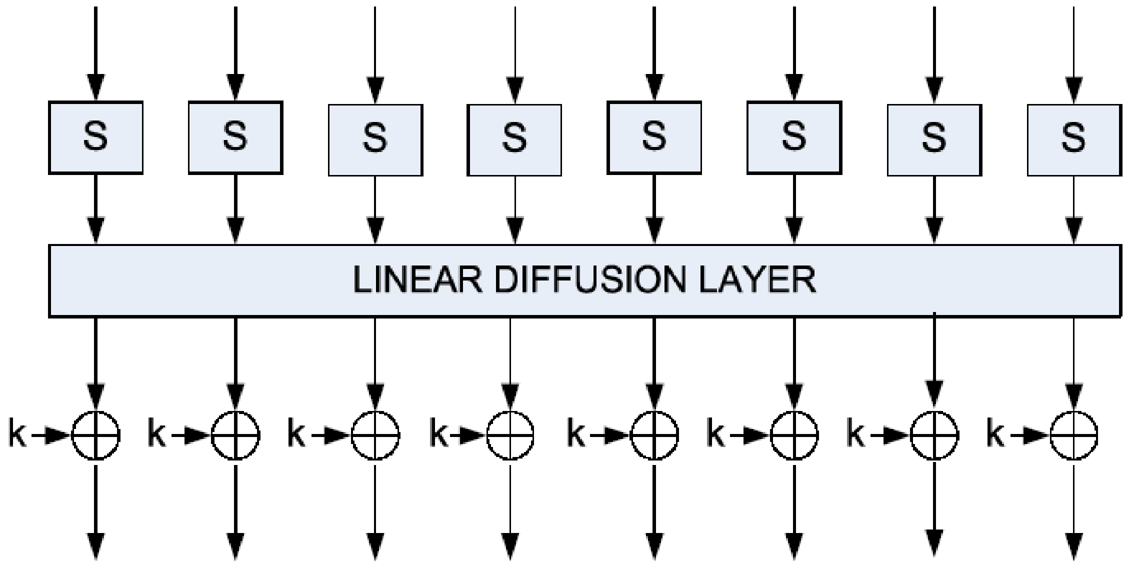
Introduction

- Countermeasures against fault attacks
  - HW, SW
  - Active, passive
- Examples:
  - Bus encryption, sensors, randomizations, …
  - Error detection techniques

$\Rightarrow$ Comparative analysis (block ciphers)

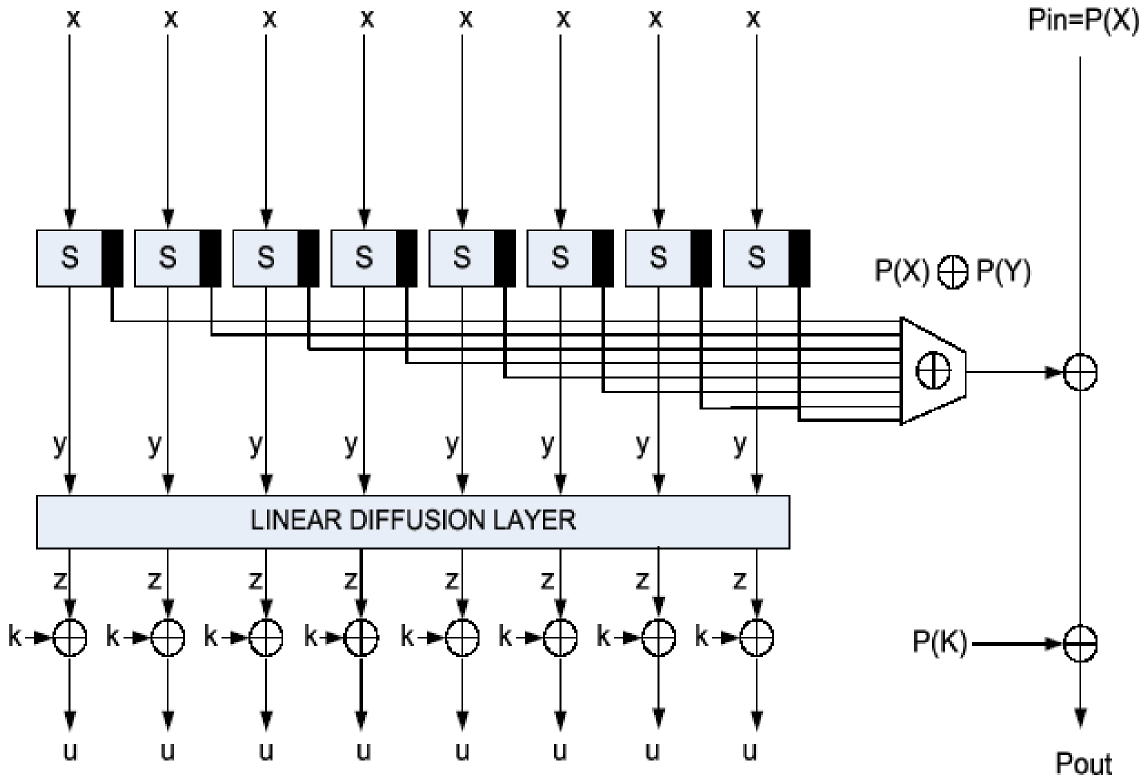
Error detection techniques using space redundancies

- Block cipher without protection:



A first proposal

Single bit parity check:



e.g. [Karri *et al*. 2003]

A first proposal

- Mainly costs an additional Boolean function for the substitution box
- Any modification of the parity will be detected at the round’s output

>< Faults of even order will *not* be detected

- OK for integrated circuits
- probably not for malicious adversaries

A first proposal

Fact 1: Probability of errors in integrated circuits

1-bit: 85%, *2-bit*: 10%, *3-bit*: 3%, *4-bit*: 1%

[Moshanin *et al*. 98]

Fact 2: Numbers of faults required to defeat, *e.g.* the AES Rijndael: 2 [Piret *et al*. 04]

Fact 3: Malicious adversaries: possibly enhanced with space and time localization

Possible improvements

- Weaknesses of the first proposal:
  - Only one parity bit is used
  - Parity codes are linear
  - (Only one checker per round)

Multiple bit parity codes

- *e.g*. [Bertoni *et al*. 03]: one parity bit per byte for the AES Rijndael
- HW penalty: the parities are now affected by the diffusion layer

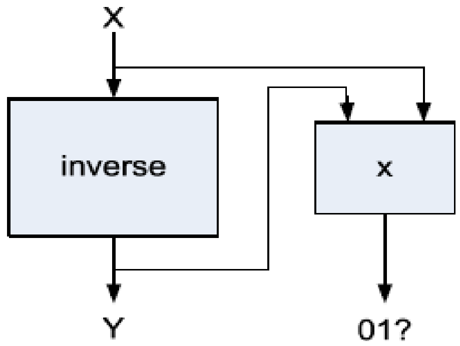
- Security improvement:

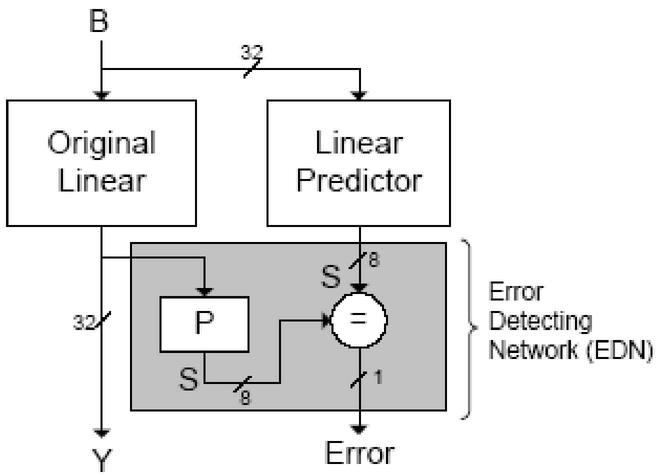
  P[double faults affecting the same byte]~12%

Non-linear robust codes #1

- [Karpovsky *et al.* 04]:
  - non-linear code for the S-box
  - check only a few bits
  - linear code for the rest:

  (8-bit parity code per column):



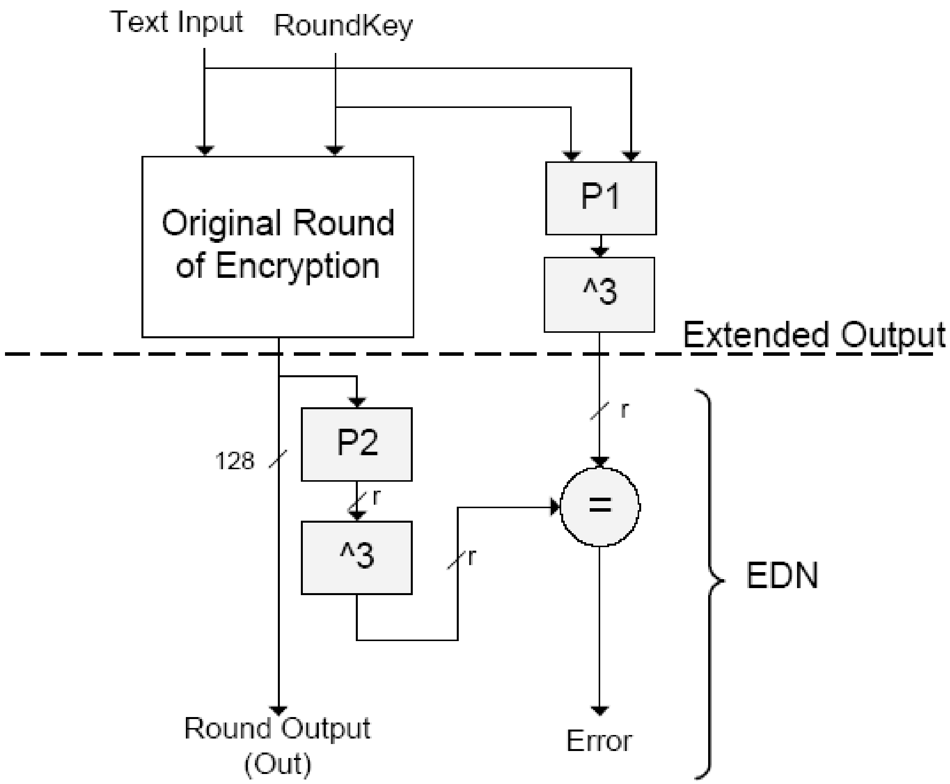


⇒ Contrasted security

Non-linear robust codes #2

- [Karpovsky *et al.* 04]:

Addition of cubic networks to the previous linear scheme:



HW cost of the different solutions

- Based on the original author's estimations

Method	Sin. fault detection	Mul. fault detection	Area overhead	Delay overhead	Thr. overhead	Thr./Area overhead
single parity bit	yes	no	+7.4%	+6.4%	-	-
multiple parity bits ($n = 16$)	yes	double faults masked with $P \propto \frac{2}{n+1}$	+20%	-	-	-
linear + non-linear codes	weak	good	+35%*	-	-	-
non-linear r-bit codes ($r = 28$)	good, missed with $P \propto 2^{-2r}$	good, missed with $P \propto 2^{-2r}$	+77%	+15%	-13%	-51%

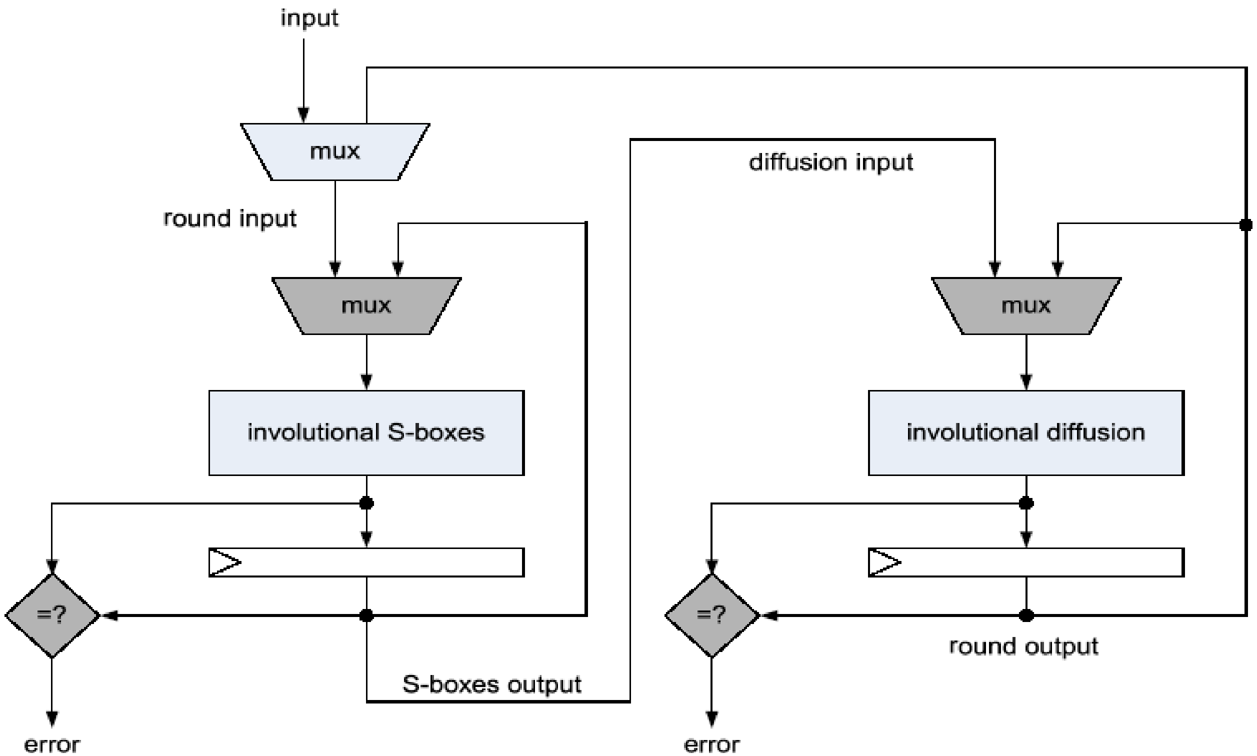
Observations

- In general, the HW overhead increases with the fault detection capabilities
- The overhead obviously depends on the cost of the original primitive (because estimated in %)

$\Rightarrow$ Security vs. efficiency tradeoff

Other proposals

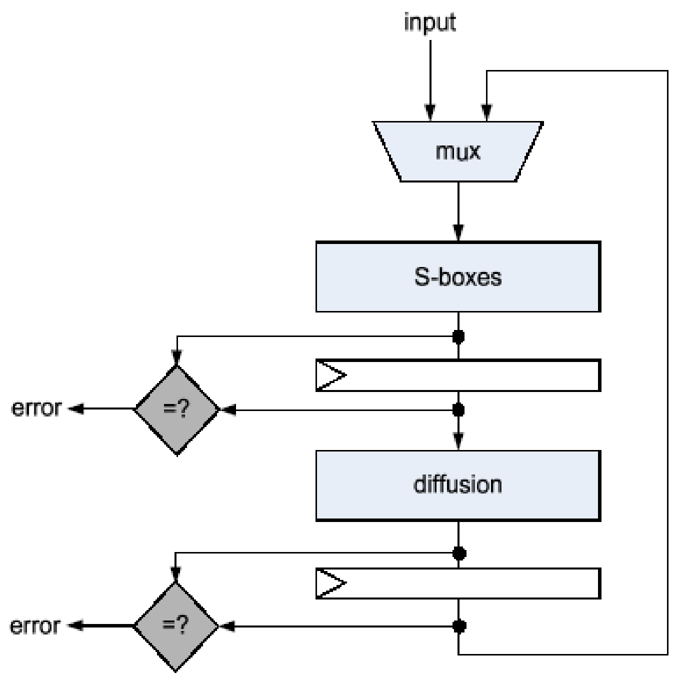
- Concurrent error detection for involution ciphers



[Joshi *et al*. 2004]

Other proposals

- What is the real cost of the proposal?
- A similar proposal would be:



$\Rightarrow$ Repetition code

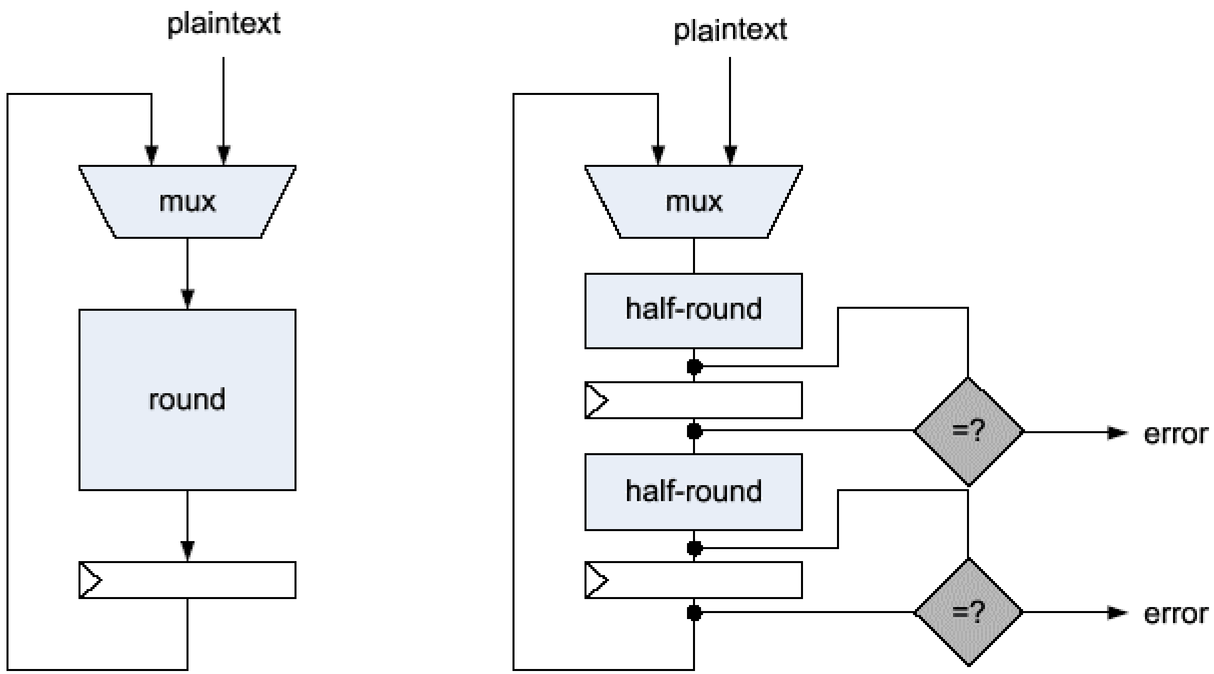
$\Rightarrow$ Throughput divided by 2

$\Rightarrow$ No permanent faults detected

Feedback modes?

- Pipeline cannot be used for efficiency…

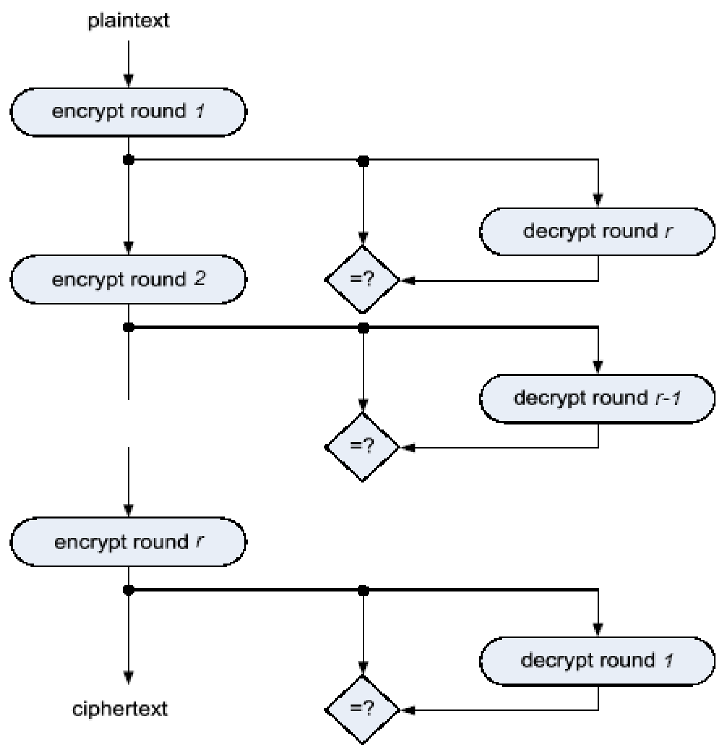
  … but can be used for fault detection



Feedback modes?

⇒ There exist contexts where fault detection can be obtained "for free"

Similar example:
[Karri *et al*. 2002]
(repetition/duplication)



Conclusions

S. Mitra, E.J. McCluskey, "*Which Concurrent Error Detection Scheme to Choose?*", International Test Conference 2000

$\Rightarrow$ Most efficient concurrent error detection schemes exceed the cost of duplication

- When can this be improved?

    Theoretically two possibilities:

  - restrict the fault model (*e.g*. multiplicities)
  - detect with lower probabilities

Both solutions are not convenient for crypto

- Or practically… in certain specific contexts:
  - Encryption in feedback modes
  - Encryption/decryption available

⇒ Purely theoretical solutions (*e.g.* algorithmic tamper proofness) are probably not completely unrealistic

? Efficiency improvements of non-linear robust codes