Using Error Detection Codes to detect fault attacks on Symmetric Key Ciphers

Israel Koren

Department of Electrical and Computer Engineering
Univ. of Massachusetts, Amherst, MA

collaborating with Luca Breveglieri, Paolo Maistri and Guido Bertoni

Department of Electronics and Information Technology
Politecnico Di Milano, Milano, Italy

Outline

- Introduction
- AES Encryption Algorithm
  - Parity code
  - Error Propagation
  - Fault Coverage
  - Fault Detecting Architecture
  - Area and Latency Overheads
  - Fault Location
- Other Symmetric Block Ciphers
  - Basic Operations
  - Error Detecting Codes
  - Frequency of Checkpoints
  - Detection Coverage
- Conclusions

Introduction

- High reliability is a desirable property of any implementation of a cryptographic algorithm
  - 1. Faults should be detected to avoid transmission of erroneous messages
  - 2. Faults may also be maliciously injected by an attacker to extract the secret key
- Fault attacks:
  - Deliberate error injection is a powerful technique that allows to retrieve the secret key with a very limited number of experiments
  - Attacks exist against AES, DES, RC4, RSA, ECC
- Conventional fault detection techniques, based on brute force redundancy, are expensive or slow

Concurrent Error Detection – Prior Approaches

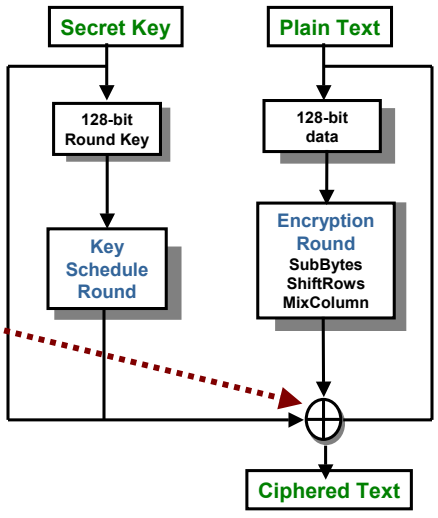
- Generic duplication
  - High area redundancy
  - Time redundancy
- Exploit unused hardware
  - Use decryption unit to check after encryption
- Exploit idle cycle time
  - Use pipelined implementations to perform repeated or inverse computations

Error Detecting Codes (EDCs)

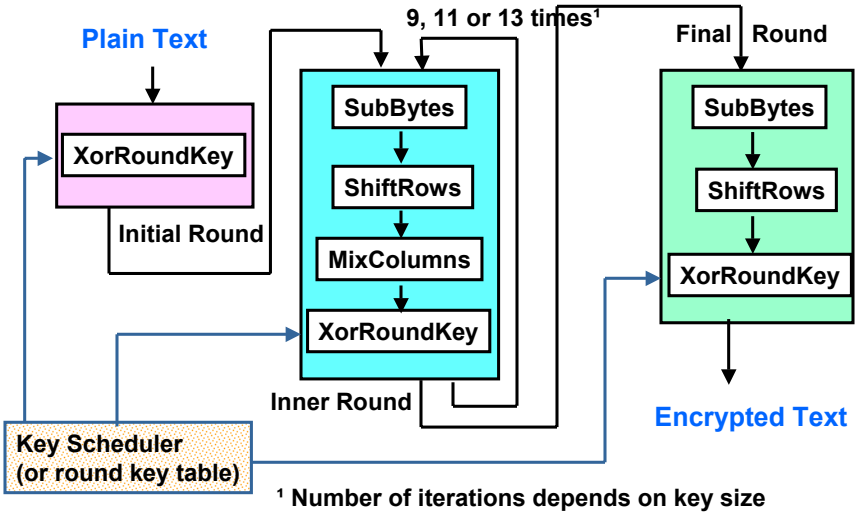
- Usually have a smaller hardware overhead than duplication
  - *Code generator*, *comparator* and *prediction circuits*
- Separable Codes:
- 1. Parity codes
  - Suited to odd-order faults
  - Allow various granularities
- 2. Residue codes
  - Require at least 2 bits (no single bit code, like parity)
  - Similar coverage for odd- and even-order faults
- Specialized codes

The AES Algorithm

- 128-bit data input
- 128 to 256-bit secret key
- Round-based:
  - **SubBytes**: Non-linear byte substitution
  - **ShiftRow**: Word rotation
  - **MixColumn**: Linear word transformation
  - **AddRoundKey**: Modulus-2 addition with round key
- **Encryption and Key Schedule use the same basic operations**
  - **Decryption uses inverse operations of encryption**

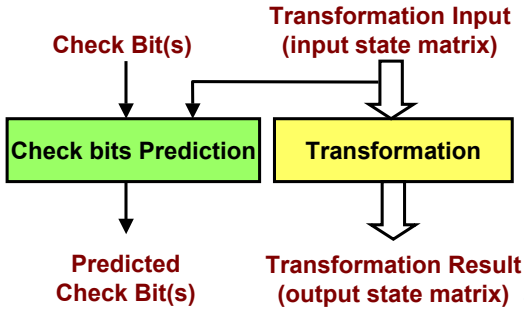


AES: Round-based Implementation

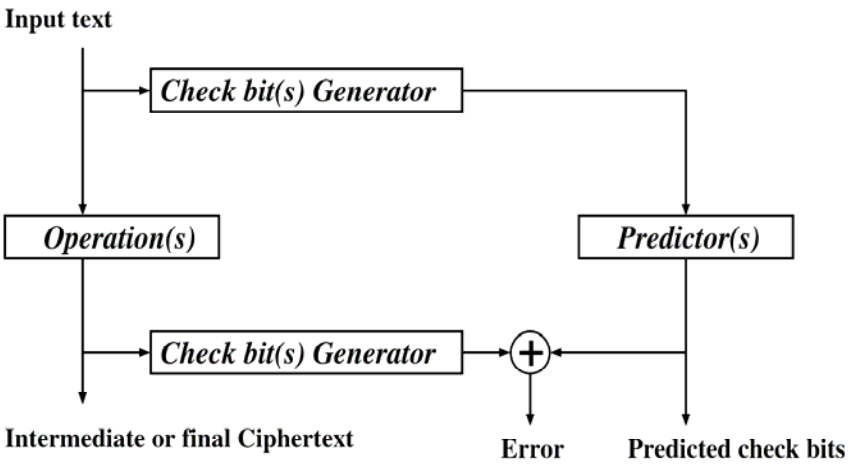


Error Detection Code (EDC) for AES

- The 128-bit data block to encrypt (or decrypt) is organized as a 4 × 4 matrix, whose entries are bytes
  - The state matrix
- The state matrix is progressively processed by the rounds and by each round internal transformation
- Check bits are associated with the state matrix, and are updated after each internal transformation
- The check bits can be verified at selected checkpoints

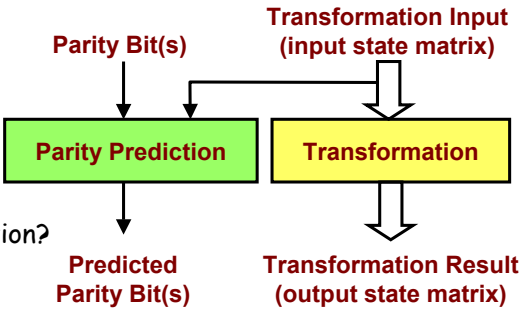


Check bits verification



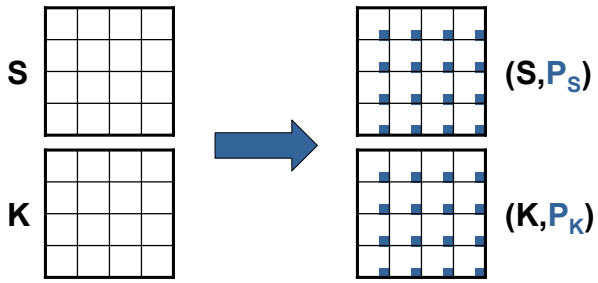
Parity Code in AES

- One parity bit is associated with each byte of the state matrix for a total of 16 parity bits
- Prediction rules are used to update the parity bits at each internal transformation
- Parity bit checkers, each with 8 inputs, compare the predicted and actual parity bits
  - **16 parity checkers per check point (8-bit each)**
- How to predict the parity bits for each transformation?



Parity bits for key matrix

- If protecting the key schedule process is necessary: A parity bit can be added to each byte of the key matrix



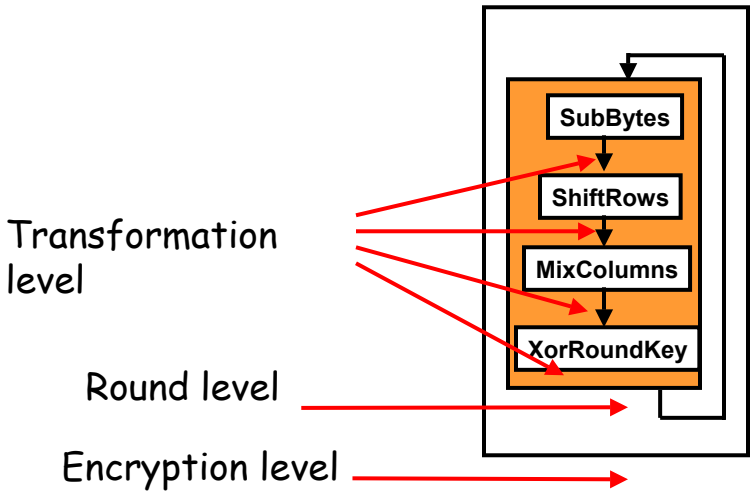
Parity Code - Prediction Rules

- **SubBytes**: implemented as a look-up table ⇒ parity bits are stored together with output data
- **ShiftRows**: parity bits are rotated to be consistent with the state
- **MixColumn**: the parity bits are updated

  e.g., $p_{0,c} = p_{0,c}+p_{2,c}+p_{3,c}+s_{0,c}^{(7)}+s_{1,c}^{(7)}$

- **AddRoundKey**: the parity bits are added (modulu-2) to the parity bits of the key

Scheduling of Check Points



Error Matrix

- If error checking after each operation is not desirable the Error Propagation must be analyzed
- Define an Error State Matrix ***E*** based on the current state and the predicted parities

$$E = P \oplus parity(S)$$

- $e_{r,c} = 1$ The byte at row r and column c is incorrect
- $e_{r,c} = 0$ Fault-free or even-order fault

Example

- A single transient fault at byte #0

Beginning of round 1	End of round 1	End of round 2
1 0 0 0	1 0 0 0	1 0 1 0
0 0 0 0	1 0 0 0	1 0 0 1
0 0 0 0	1 0 0 0	1 0 1 1
0 0 0 0	0 0 0 0	0 0 1 1

Rounds from 3^{rd} to 7^{th}	End of Round 7	End of Round 8
.....	1 0 0 0	1 0 0 0
.....	0 0 0 0	0 0 0 0
.....	0 0 1 0	0 0 0 0
.....	0 0 0 1	0 0 0 0

The Z Matrix

- Error in datapath propagates as $E = ZE$

For n rounds $E = Z^n E$

$$\begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0
\end{pmatrix}$$

Error Propagation Model

- The 16×16 matrix Z, $z_{i,j} \in GF(2)$, depends on the round operations
- Error can be detected as long as the error matrix E is not completely zeroed
- Z is orthogonal, thus it never completely cancels an error

Extended Error Propagation Model

- If Key Schedule is also vulnerable, the model must be extended

$$E_K^{(r+1)} = Z_K E_K^{(r)}$$

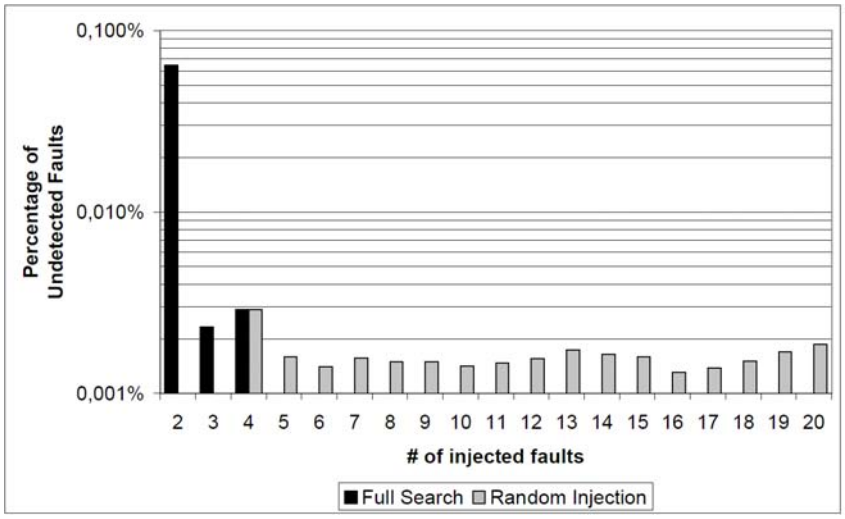
$$E^{(r+1)} = Z E^{(r)} + E_K^{(r)}$$

- The matrix Z_K models error evolution in the Key Schedule
  - It is non singular, but only Z^{15} is orthogonal and Z^{60} is the identity matrix
  - AES has at most 14 iterations

Error Coverage

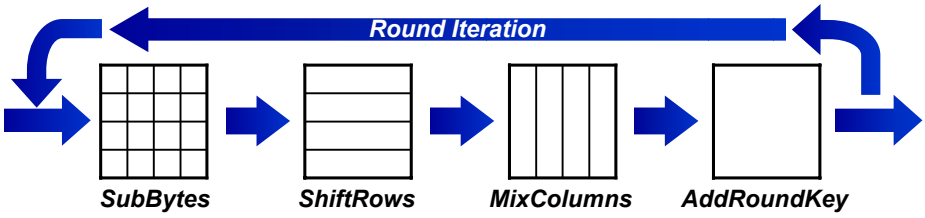
- A single checkpoint scheduled at the end of the encryption process
- All single faults are detected (analytical proof)
- Multiple fault coverage evaluated through simulation experiments - Faults injected randomly
- Double faults not detected if:
  - The two faults affect the same byte simultaneously
  - The two faults hit the same byte, but are separated by a distance of 8 rounds (since error propagation is periodic over 8 rounds)
- The total double fault coverage is 99.125%

Error Coverage – simulation results

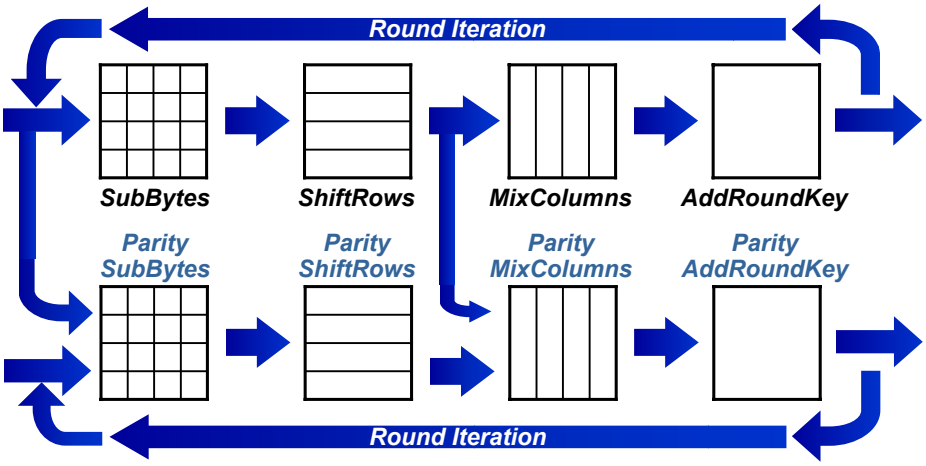


Hardware Implementation - Example

- Basic reference architecture
  - No loop unrolling
  - No pipelining
  - No decryption
- Full 128-bit datapath width
  - Parallel computation on entire block
  - 16 S-Boxes
  - 4 MixColumns in parallel



Extended Architecture



Error Detection Overheads

- Adding error detection increases area and latency
- Larger area
  - Each operation is extended - check bits prediction
  - Code generators are needed for code initialization
  - Code comparator needed at the end to check for errors
- Higher latency
  - Initial code generation and final validation need sequential computation

Overheads - results

Design I (S-Box)	Area (μm^2)	Latency (*ns*)	AT
Base	233,095	8.88	2,069,883
w/Error Detection	271,245	11.06	2,999,969
Overhead	+16.36%	+24.55%	+44.93%

Design II (S-Box)	Area (μm^2)	Latency (*ns*)	AT
Base	226,099	9.32	2,107,242
w/Error Detection	259,631	12.27	3,185,672
Overhead	+14.83%	+31.62%	+51.18%

Implementation Summary

- VHDL can be easily extended to include error detection capabilities
  - Area overhead is about 15 - 16%
  - Latency overhead is between 24 - 32%
- Latency overhead is mainly due to the code comparator
  - Can be reduced by moving the comparator out of the critical path
- Common design improvements can be followed
  - E.g., pipelining, loop unrolling, …

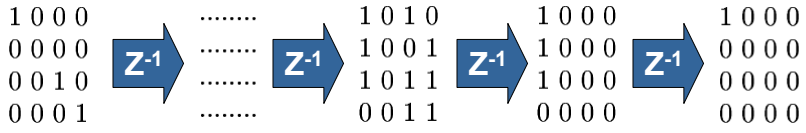
Fault Location - Goals

- Higher reliability
  - e.g., have a redundant byte slice and connect it upon locating a permanently faulty byte slice
- Possibly better protection against fault injection attacks
  - e.g., byte parity is unable to detect all injected errors
  - upon a detected fault, replace the byte currently under attack (e.g., by a laser beam) by a spare

Fault Location – Byte Parity

- 8 rounds after the fault injection, the error state matrix restores its initial value
  - Any attempt to locate the fault (byte and round) must be within 8 rounds
- For each Single Transient Fault (STF) in a byte, the error state matrix is unique at each round (within the 8 rounds limit)
  - For two different STFs the corresponding error state matrices are different
- Given an admissible error state matrix, we can identify the original STF
  - 1. By scanning a look-up table
  - 2. By multiplying the error state matrix by $\mathbf{Z}^{-1}$ until we find an STF

Identifying Single Transient Faults



Error state matrix after 8 rounds	Fault injected at the beginning of round			
Byte	1	2	...	8
0	1	33793	...	7
...	...	...	...	...
15	32768	2114	...	11

- Unique fault signatures ⇒ All single transient faults can be located

Single Permanent Faults (SPFs)

- An SPF is a stuck-at fault affecting always the same bit
- Depending on the data input and the key, the SPF may manifest itself in any number of rounds out of the 8
- Fault location must use look-up tables
  - Backward computation is not feasible due to the excessive number of fault manifestations

Single Permanent Fault

- SPF Signature matrix
  - Faulty byte (from 0 to 15)
  - Temporal manifestation (2^8-1 = 255 configurations)
  - 4080 signature values

	0	1	2	3	4	5	6	7	8	9	...	15
0x01	1	2	4	8	16	32	64	128	256	512	...	32768
0x02	33793	32801	1057	33824	16408	536	16912	16904	388	8576	...	2114
0x03	33792	32803	1061	33832	16392	568	16976	17032	132	9088	...	34882
...	...	...	...	...	...	...	...	...	...	...	...	...
0x11	43947	22359	44718	23901	47802	30069	60138	54741	43947	22359	...	54741
...	...	...	...	...	...	...	...	...	...	...	...	...
0x77	15163	56540	52942	29555	46003	52685	60652	14135	15163	56540	...	14135
0x78	16425	30958	14510	50863	660	36583	35555	27388	10560	61048	...	64618
0x79	16424	30956	14506	50855	644	36551	35491	27260	10304	60536	...	31850
...	...	...	...	...	...	...	...	...	...	...	...	...
0x87	10560	61048	44600	44998	37890	59278	58250	64618	16425	30958	...	27388
...	...	...	...	...	...	...	...	...	...	...	...	...
0xFF	26985	38550	38550	26985	38550	26985	26985	38550	26985	38550	...	38550

4080 Single Permanent Fault Signatures

- 3072 are unique
- 1008 (<25%) are not unique - two or more entries have an identical value
- Solution: Use a 2nd run

	0	1	2	3	4	5	6	7	8	9	...	15
0x01	1	2	4	8	16	32	64	128	256	512	...	32768
0x02	33793	32801	1057	33824	16408	536	16912	16904	388	8576	...	2114
0x03	33792	32803	1061	33832	16392	568	16976	17032	132	9088	...	34882
...	...	...	...	...	...	...	...	...	...	...	...	...
0x11	43947	22359	44718	23901	47802	30069	60138	54741	43947	22359	...	54741
...	...	...	...	...	...	...	...	...	...	...	...	...
0x77	15163	56540	52942	29555	46003	52685	60652	14135	15163	56540	...	14135
0x78	16425	30958	14510	50863	660	36583	35555	27388	10560	61048	...	64618
0x79	16424	30956	14506	50855	644	36551	35491	27260	10304	60536	...	31850
...	...	...	...	...	...	...	...	...	...	...	...	...
0x87	10560	61048	44600	44998	37890	59278	58250	64618	16425	30958	...	27388
...	...	...	...	...	...	...	...	...	...	...	...	...
0xFF	26985	38550	38550	26985	38550	26985	26985	38550	26985	38550	...	38550

Multiple Transient Faults (MTF)

- MTFs are the most general including both STFs and SPFs as special cases
- Location of MTFs can be misleading
  - The same error state matrix can be obtained using several different sets of MTFs
  - E.g.: Signature matrix due to two transient faults has 384 values overlapping with the SPF matrix
    - I.e., if one of such pairs occurs, the parity code may misidentify it as an SPF

Detecting Faults in other Symmetric Key Block Ciphers

- Evaluate error detecting codes for other ciphers which use different basic operations, e.g.,
  - Bit-oriented operations (DES)
  - Modular arithmetic with unusual modulus (IDEA)
- Provide guidelines for choosing the best EDC, given a set of operations
- Suggest the best code granularity
  - How many check bits, and where to be placed?
- Determine the best checkpoint frequency
  - When is checking mandatory?
- Evaluate the detection coverage

Symmetric Block Ciphers - Operations

Ciphers	XOR	+, -	x	Sbox	Rot	Perm	x mod G(x)
DES	✓			✓		✓	
IDEA	✓	✓	✓*			✓	
MARS	✓	✓	✓	✓	✓	✓	
RC5	✓	✓			✓		
RC6	✓	✓	✓		✓	✓	
Rijndael (AES)	✓			✓		✓	✓
Twofish	✓	✓		✓	✓	✓	✓

- **Comments**: XOR: always used
- ADD: often used
  - **SUB: in encryption is used only in MARS**
- Integer and polynomial multiplication appear in recent ciphers
- S-Boxes needed for non-linearity
  - **Rotation or modular multiplication as alternative**
- Rotation can be even data-dependent
- Permutations provide *scrambling*

Feasibility of EDCs - Operations

Operation	Parity	Residue
XOR	Yes	Yes
Integer Add, Subtract	Yes	Yes
Integer Multiplication	Expensive	Yes
Substitution Box	Yes	Expensive
Rotation	Yes	Yes
Permutation	Yes	Yes
Polynomial Multiplication	Yes	Expensive

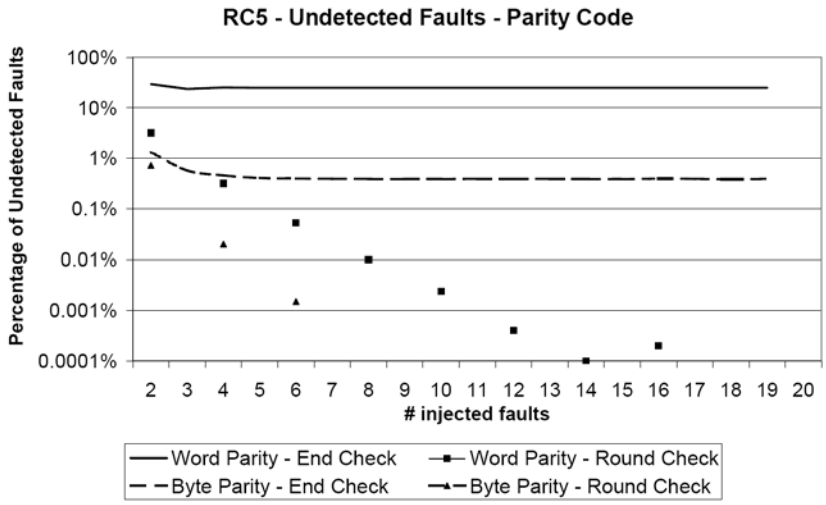
- **Comments**: 1. Integer operations require a correction term due to possible overflow
- 2. Parity prediction in integer multiplication must consider all internal carries, which may not always be available
- 3. Rotations and permutations at the same level of code are trivial
  - The check bits move along with the data

Preferred EDCs

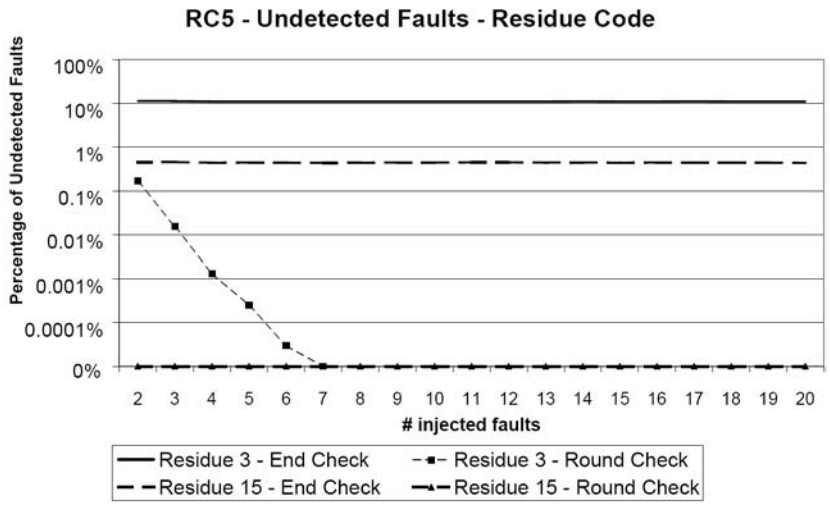
Cipher	Suggested Code
DES	Parity
IDEA	Residue, but expensive
MARS	Residue, but expensive
RC5	Parity or residue
RC6	Residue
Rijndael (AES)	Parity, per byte
Twofish	Parity, per byte

- **Comments:**
- AES uses polynomial multiplication, hence parity is preferred
- RC5 uses both logical and arithmetic operations - suited to both codes
- DES uses small tables, which can be easily extended with parity bits
- IDEA uses multiplication mod $(2^{16}+1)$ - must use expensive residue codes
- Similar guidelines for other ciphers (MARS, RC6, Twofish)

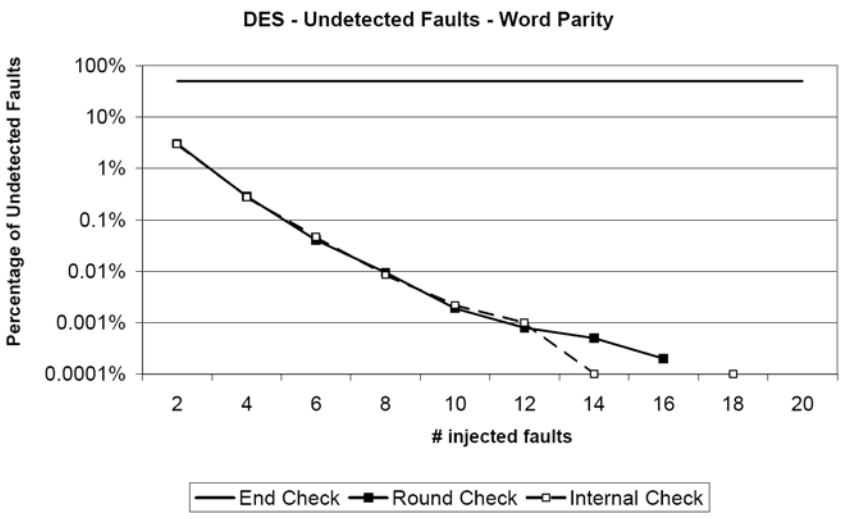
Detection Coverage RC5 (Parity)



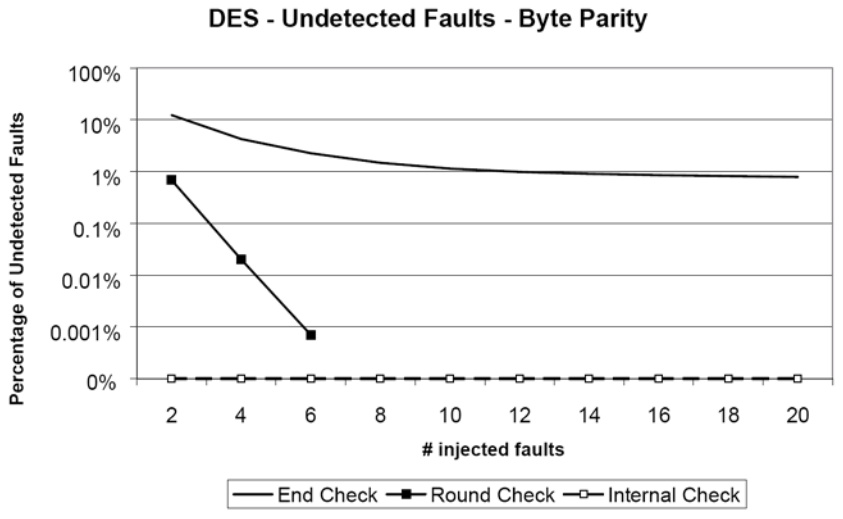
Detection Coverage RC5 (Residue)



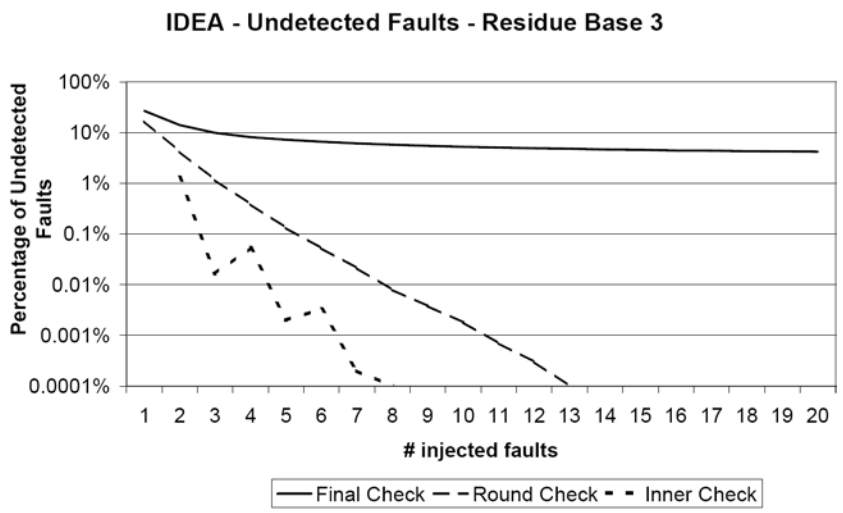
Detection Coverage
DES (Word Parity)



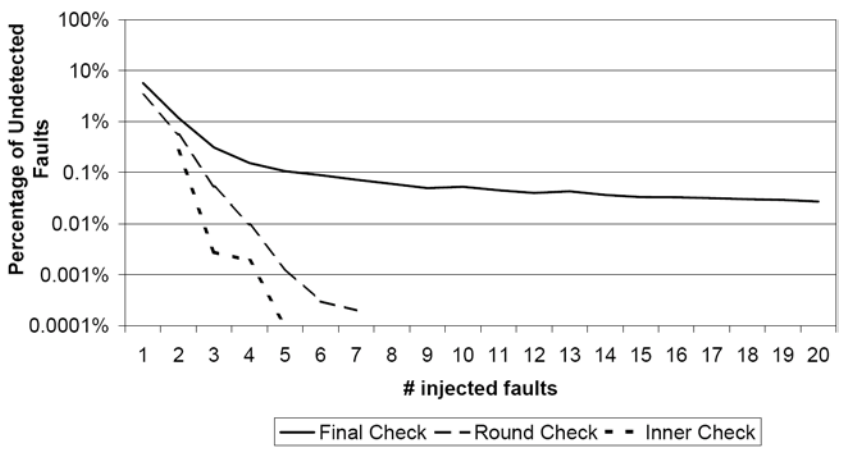
Detection Coverage
DES (Byte Parity)



Detection Coverage
IDEA (Residue Mod 3)



Detection Coverage
IDEA (Residue Mod 15)



Conclusions

- Error detecting codes are appropriate for a wide range of cryptographic operations
  - The ciphers analyzed here cover logical, arithmetic and modular operations
  - Not covered are logical AND and OR, which are cheaper to protect by means of duplication (e.g., Camellia)
- Many degrees of freedom
  - Type of code (parity, residue, etc.)
  - Frequency of checkpoints
  - Number of check bits (i.e., amount of redundancy)
- Future directions:
  - Analyze other EDCs
  - Develop library of cryptographic functional units supporting concurrent error detection by EDCs
  - Develop reconfigurable fault tolerant designs

Publications - 1

- L. Breveglieri, I. Koren and P. Maistri, "An Operation-Centered Approach to Fault Detection in Symmetric Cryptography Ciphers," to appear, IEEE Trans. on Computers, 2007.
- G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," IEEE Trans. on Computers, pp. 492-505, April 2003.
- L. Breveglieri, I. Koren and P. Maistri, "A Fault Attack Against the FOX Cipher Family," Proc. of FDTC 2006, LNCS, Vol. 4236, pp. 98-105, Springer-Verlag, Oct. 2006.
- L. Breveglieri, I. Koren and P. Maistri, "Incorporating Error Detection and Online Reconfiguration into a Regular Architecture for the Advanced Encryption Standard," Proc. of DFT'05, pp. 72-80, Oct. 2005.
- L. Breveglieri, I. Koren, P. Maistri and M. Ravasio, "Incorporating Error Detection in an RSA Architecture," LNCS, Vol. 4236, pp. 71-79, Springer-Verlag, Oct. 2006.
- L. Breveglieri, I. Koren and P. Maistri, "An Efficient Hardware-Based Fault Diagnosis Scheme for AES: Performances and Cost," Proc. of DFT'04, pp. 130-138, Oct. 2004.
- L. Breveglieri, I. Koren and P. Maistri, "Detecting Faults in Four Symmetric Key Block Ciphers," Proc. of ASAP'04, pp. 258-268, Sept. 2004.

Publications -2

- L. Breveglieri, I. Koren and P. Maistri, "Detecting Faults in Integer and Finite Field Arithmetic Operations for Cryptography," Proc. of FDTC'04, pp. 361-367, June 2004.
- G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Detecting and Locating Faults in VLSI Implementations of the Advanced Encryption Standard," Proc. of DFT'03, pp. 105-113, November 2003.
- G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Concurrent Fault Detection in a Hardware Implementation of the RC5 Encryption Algorithm," Proc. of ASAP'03, pp. 423-432, June 2003.
- G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "A Parity Code Based Concurrent Fault Detection for Implementations of the Advanced Encryption Standard," Proc. of DFT'02, pp. 51-59, November 2002.
- G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "On the Propagation of Faults and Their Detection in a Hardware Implementation of the Advanced Encryption Standard," Proc. of ASAP'02, pp. 303-312, July 2002.
- G. Bertoni, L. Breveglieri, I. Koren and V. Piuri, "Fault Detection in the Advanced Encryption Standard," Proc. of MPCS'02, pp. 92-97, April 2002.