A Novel Double-Data-Rate AES Architecture Resistant against Fault Injection

P. Maistri, P. Vanhauwaert, R. Leveugle
TIMA Laboratory – Grenoble – FRANCE
Outline

- Motivation and objectives
- Current detection countermeasures
- The DDR approach: pros and against
- The AES implementation
  - The reference design
  - DDR issues: alignment and synchronization
  - Operation modes
- Robustness evaluation
- Conclusions
Motivation

- Fault attacks are one of the most effective ways to break a cryptosystem
  - AES can be broken with 2 well-located faults (Piret-Quisquater, CHES 2003)
- Offline error detection can not guarantee enough protection against the attacks
- Current detection countermeasures are expensive and/or have poor efficiency against realistic attacks
- The error detection scheme must be efficient against both natural and intentional faults
Concurrent Error Detection Schemes

- Based on spatial redundancy:
  - Circuit duplication

- Based on information redundancy:
  - Error detecting codes: parity (Bertoni et al. TC 2003), non-linear cubic codes (Karpovsky et al., DSN 2004)

- Based on temporal redundancy:
  - Computation of the inverse process (e.g., decryption) with additional (possibly existing) hardware (Karri et al., 2001)
  - Computation of the inverse process with the same hardware, for involution ciphers only (Joshi et al., CHES 2004)
  - Repetition of the same process, exploiting a pipeline (Wu and Karri, DFT 2001)
What is not good so far...

- Error codes for AES are either expensive (non-linear networks) or inefficient against malicious faults (parity)
- Spatial/information redundancy may increase correlation with power consumption and EM emissions, thus favoring side-channel attacks
- Temporal redundancy:
  - Process repetition involves performance overhead
  - Pipeline implementation requires fast system clock and significant area overhead (+50%), but ...
  - ... the global system may work at reduced frequency, thus affecting the global throughput
Double-Data-Rate Computation

- Twice the throughput at the same frequency
- Small area overhead for DDR logic
- Increased parallelism

- More complex routing, thus lower max frequency
- Error detection requires additional overhead
- Design may require synchronization “bubbles”

4 clock cycles to compute Operation for all input bytes

2 clock cycles to compute Operation for all input bytes
AES – The architecture

- 32-bit data-path
- 4 Substitution Boxes
- 16 GF Multipliers for MixCol
- 3 clock cycles per round
- On-the-fly key unrolling

- MixColumns, AddRoundKey and State
- 2-stage SBox
- Register layer
- Combinatorial logic
- 8-bit signal
- 32-bit signal
Data Alignment in AES

- The data alignment phase partitions the register space into two classes:
  - Registers triggered by ascending clock edge
  - Registers triggered by descending clock edge
- Alignment can be done:
  - By columns: registers in the same columns share the clock alignment
  - By rows: registers in the same rows share the clock alignment
  - By checkers: elements of the partitions are interleaved both in columns and rows, like a chess board
Synchronization

- DDR computation can be employed when we have scarce resources, high parallelism and no data dependency
  - In our design, SBoxes are the scarce resources
  - Row rotation is performed while moving data during non-linear substitution (collateral data-dependence)
  - Row-wise DDR alignment is thus chosen

- In AES, all operations are independent on each byte, but the \textit{MixColumns} operation
  - \textit{MixColumns} are not a scarce resource (each byte is computed locally), but values have to be stable (i.e., a latch is used)
Operation modes

- **Single**: the unit uses the DDR computation to improve its throughput and no check is performed on data.

- **Double**: the unit uses the DDR computation to compute each round twice, checking for inconsistencies.

- **Interleaved**: like the *Double* mode, but the first and second repetition process two different (consecutive) blocks in ECB mode, sharing the encryption key.

\[
\begin{align*}
  & (\text{dummy, } P_1) & (P_1, P_2) & (P_2, P_3) & (P_3, P_4) & (P_4, P_5) \\
  & (\text{dummy, } C_1) & (C_1, C_2) & (C_2, C_3) & (C_3, C_4) & (C_4, C_5)
\end{align*}
\]
## Cost Comparison

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Notes</th>
<th>Area Overhead</th>
<th>Throughput Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Parity Bits</td>
<td>One parity bit per byte, expensive SBox protection</td>
<td>33%</td>
<td>3%</td>
</tr>
<tr>
<td>Bertoni et al., TC ’03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inverse Process</td>
<td>Dec after enc at block, round or operation level</td>
<td>19% - 38%</td>
<td>23% - 61%</td>
</tr>
<tr>
<td>Karri et al., DAC ’01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipeline Recomputation</td>
<td>Uses unused stages to redo computation in RC6</td>
<td>50%</td>
<td>18%</td>
</tr>
<tr>
<td>Wu and Karri, DFT ’01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Parity Bit</td>
<td>One parity bit per block, aimed at stuck-at faults</td>
<td>18% - 24%</td>
<td>NA</td>
</tr>
<tr>
<td>Karri et al., CHES ’03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-linear Code</td>
<td>Non-linear scalable cubic network</td>
<td>77%</td>
<td>13%</td>
</tr>
<tr>
<td>Karpovsky et al., DSN ’04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDR</td>
<td>Suitable for fast designs in slower systems</td>
<td>36%</td>
<td>15% - 55%</td>
</tr>
</tbody>
</table>
Fault Injection

- Fault injection was based on hardware emulation
- Injection software ran on the FPGA PowerPC
  - Reduced communication, thus faster execution of the campaign due to less wasted time
  - Load can be distributed at any level: hw logic, FPGA PPC, host
- Extra logic is added to the original AES description
  - For each targeted flip-flop, one XOR is inserted between the FF and the combination block at its input
## Error Detection

<table>
<thead>
<tr>
<th>Instrumented Target</th>
<th>Result Class [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Location</td>
</tr>
<tr>
<td>Protected targets:</td>
<td></td>
</tr>
<tr>
<td>Linear layer</td>
<td>16*</td>
</tr>
<tr>
<td>SBox Output</td>
<td>16*</td>
</tr>
<tr>
<td>Inner SBox</td>
<td>24</td>
</tr>
<tr>
<td>Non protected targets:</td>
<td></td>
</tr>
<tr>
<td>Misc ctrls</td>
<td>22</td>
</tr>
<tr>
<td>Key ctrls</td>
<td>3</td>
</tr>
<tr>
<td>Main FSM</td>
<td>19</td>
</tr>
<tr>
<td>Aux FSM</td>
<td>9</td>
</tr>
<tr>
<td>FSM Synchr</td>
<td>6</td>
</tr>
</tbody>
</table>

* Full search on single byte (8-bit target) gave the same results
## Coverage Comparison

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Area Overhead</th>
<th>Throughput Reduction</th>
<th>Coverage Byte Errors in Datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Parity Bits</td>
<td>33%</td>
<td>3%</td>
<td>~67%</td>
</tr>
<tr>
<td>Bertoni et al., TC ’03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inverse Process</td>
<td>19% - 38%</td>
<td>23% - 61%</td>
<td>100%</td>
</tr>
<tr>
<td>Karri et al., DAC ’01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipeline Recomputation</td>
<td>50%</td>
<td>18%</td>
<td>~100%</td>
</tr>
<tr>
<td>Wu and Karri, DFT ’01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Parity Bit</td>
<td>18% - 24%</td>
<td>NA</td>
<td>~67%</td>
</tr>
<tr>
<td>Karri et al., CHES ’03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-linear Code</td>
<td>77%</td>
<td>13%</td>
<td>~100%</td>
</tr>
<tr>
<td>Karpovsky et al., DSN ’04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDR</td>
<td>36%</td>
<td>15% - 55%</td>
<td>~100%</td>
</tr>
</tbody>
</table>
Vulnerabilities

- DDR applies to data path only, control unit must be addressed with other protection means
  - Protection of the control unit is envisioned in a more recent version, exploiting selected duplication, transition verification, state validation

- Coverage of the data path is not 100% for multiple-bit errors
  - A small percentage (0.06%) of errors injected into the inner registers of SBoxes is not detected: this issue is currently under investigation

- Permanent fault may not be detected
  - They are outside the scope of this work, which is focused on transient faults (either natural or intentional)

- Tailored attacks are not detected
  - The attacker must be able to inject the same error value in the same location at very specific time slots: very difficult and unlikely with current attack capabilities
Conclusions

- The DDR approach is an alternative computation template to improve computation parallelism with scarce resources.
- Like other solutions, more complex routing implies lower maximum frequency...
- … but embedded in slower-clock systems it may double the throughput, or allow error detection by recomputation.
- Coverage of short (one-cycle) multiple-bit errors in the data path is almost 100%.
- Attacks are possible if the same error is injected twice at specific time slots, which is unlikely:
  - The attacker can finely control the injected error value.
  - The second error value is equal to the first one by chance.
  - Errors are due to permanent faults.