Fault Attack Resistance Using Intra-Instruction Redundancy

Conor Patrick, Bilgiday Yuce, Nahid Farhady Ghalaty, Patrick Schaumont

Secure Embedded Systems, Virginia Tech

This research was supported in part by the NSF Grant 1441710, Semiconductor Research Corporation Task 2552.001, and NSF CyberCorps Program.

August 12, 2016
This presentation

- Secure software countermeasure against fault attacks

1. Why fault attacks
2. Current countermeasures
3. Intra-Instruction Redundancy (IIR)
4. Improve upon IIR
5. Results
Fault Attacks

- Method for getting secrets or processor control
- S. Ali et. al found that AES can be broken with just two fault injections

Fault attacks need two things

- Ability to **inject** fault
- Ability to **observe** there was fault (this is what countermeasures focus on)
Fault attack countermeasures

- All leverage some form of redundancy
  - Error correcting codes, duplicated execution
  - Can be in hardware or software
- Or detectors
  - Clock or voltage glitch detectors, temperature sensors
  - Requires special hardware
Our motivation

- Hardware solutions are expensive and slow to market
- Can we resist fault attacks using only software?
Software countermeasures: **Algorithm Duplication**
Software countermeasures: Algorithm Duplication

Detected!

Intra-Instruction Redundancy. Conor Patrick.
Software countermeasures: Algorithm Duplication

Plaintext → Block encryption 1 → Block encryption 2 → Continue?

Time

Not detected!

Instra-Instruction Redundancy. Conor Patrick.
Software countermeasures: **Instruction Duplication**

- Instruction A
- Instruction A’
- Test A == A’
- Test A == A’

Encryption

Plaintext → Ciphertext

Time

Intra-Instruction Redundancy. Conor Patrick.
Software countermeasures: Instruction Duplication

Instruction A
Instruction A'
Test A == A'
Test A == A'

Detecting Intra-Instruction Redundancy.

Plaintext           Encryption           Ciphertext

Time

Detected!
Software countermeasures: Instruction Duplication

Intra-Instruction Redundancy. Conor Patrick.
Software countermeasures: Instruction Duplication

Check out our paper:
Software Fault Resistance is Futile: Effective Single-glitch Attacks

Not Detected!

Intra-Instruction Redundancy. Conor Patrick.
Software countermeasures: **Infective**

All attempts have been broken

Instra-Instruction Redundancy. Conor Patrick.
Intra-Instruction Redundancy (IIR)

- Redundancy is not separated by time
- Generic to any bit-sliceable algorithm (block ciphers)
- Can integrate with other countermeasures
Our software countermeasure: Intra-Instruction Redundancy (IIR)
Our software countermeasure: Intra-Instruction Redundancy (IIR)

An adversary must make a target 2 bit fault in a processor word

Intra-Instruction Redundancy. Conor Patrick.
How to implement? With bit-slicing.

- 32 bit processor word
- 32, 128-bit blocks to encrypt

Transpose

Intra-Instruction Redundancy. Conor Patrick.
IIR Slice Allocation

Known ciphertext slices

bits N

bits N+1

bits N+2

bits N+3

redundant slices

B3'  B3  B2'  B2  B1'  B1  B0'  B0

128 words

Intra-Instruction Redundancy. Conor Patrick.
Theoretical Fault Coverage

- Random word: ~100%
- Random byte: 94.90%
- Random bit: 100%
- Instruction skip: 75%
- Chosen pair: 51.61%
Problem: rounds are time separated

Intra-Instruction Redundancy. Conor Patrick.
Solution: make each slice a different round
Improving IIR by adding Pipelining

Intra-Instruction Redundancy. Conor Patrick.
Theoretical Fault Coverage

- Random word: ~100%
- Random byte: 99.90%
- Random bit: 100%
- Instruction skip: 99.90%
- Chosen pair: 96.77%
More: add random shifts

Intra-Instruction Redundancy. Conor Patrick.
Experimental Results Setup

- We tested our countermeasures in simulation
  - 32 bit SPARC/LEON3 simulator by Cobham Gaisler
  - Gives cycle accurate performance measurements
  - Wrote a wrapper program to extend it to simulate various fault scenarios

- Ran fault tests on the SBOX part of a AES implementation we wrote
- Each simulation injected 20,200 data faults and 7,200 instruction skips.
Our reference bit-sliced AES Implementation

- Implemented our own bit-sliced AES
- Made 3 forks of it to test 3 different countermeasures

32 bit SPARC/LEON3 overhead:

<table>
<thead>
<tr>
<th>Performance</th>
<th>Program size</th>
</tr>
</thead>
<tbody>
<tr>
<td>469.3 cycles/byte</td>
<td>5576 bytes</td>
</tr>
</tbody>
</table>

* This is slow but relative performance of countermeasures will scale with performance of base implementation
## Countermeasure Overhead

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Footprint</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unprotected AES</strong></td>
<td>469.3 cycles/byte</td>
<td>5576 bytes</td>
</tr>
<tr>
<td><strong>IIR-AES</strong></td>
<td>1055.9 cycles/byte</td>
<td>6357 bytes</td>
</tr>
<tr>
<td><strong>Pipelined IIR-AES</strong></td>
<td>1942.9 cycles/byte</td>
<td>5688 bytes</td>
</tr>
<tr>
<td><strong>Shuffled Pipelined IIR-AES</strong></td>
<td>1957 cycles/byte</td>
<td>6134 bytes</td>
</tr>
</tbody>
</table>

*Instra-Instruction Redundancy. Conor Patrick.*
Countermeasure Program Size Overhead

Program Size

<table>
<thead>
<tr>
<th></th>
<th>4500</th>
<th>5000</th>
<th>5500</th>
<th>6000</th>
<th>6500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unprotected AES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIR-AES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelined IIR-AES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shuffled Pipelined IIR-AES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

bytes

by 

bytes

Intra-Instruction Redundancy. Conor Patrick.
Countermeasure Performance Overhead

Performance Overhead

- Cycles/byte

Unprotected AES | IIR-AES | Pipelined IIR-AES | Shuffled Pipelined IIR-AES

Intra-Instruction Redundancy. Conor Patrick.
Experimental Results

- Random word: 100%
- Random byte: 99.99%
- Random bit: 100%
- Instruction skip: 98.86%
- Chosen pair: 98.6%
To conclude

- Introduced a novel method for software fault detection using IIR
  - We believe this is the best you can do to protect from faults in SW

- Protect from well targeted, repeatable faults.
- Acceptable performance costs and minimal program size overhead.
- Verified our fault coverage in simulation.
Thank you

Questions?

Web: www.faculty.ece.vt.edu/schaum/research/

Twitter: @_conorpp

Email: conorpp@vt.edu

Github: Secure-Embedded-Systems