UCI University of California, Irvine

CAMFAS: A *C*ompiler *A*pproach to *M*itigate *F*ault *A*ttacks via Enhanced *S*IMDization

Zhi Chen[1], **Junjie Shen**[1], Alex Nicolau[1], Alex Veidenbaum[1]

Nahid Farhady Ghalaty[2] and Rosario Cammarota[3]

[1]University of California, Irvine

[2]Accenture Cyber Security Technology Labs, Virginia, USA

[3]Qualcomm Research, San Diego, USA

QUALCOMM®

accenture
High performance. Delivered.

Fault Attacks

- Fault attacks occur as intentional disturbance of a micro-processor
  - To exploit the secret keys of crypto modules.
  - To take control of the micro-processor actions.

Fault Attack Process

- Fault attack vectors include two main steps:
  - Fault measurement – the process to get faulty data
    - Fault Injection.
    - Fault effect observation.
  - Fault analysis – techniques to process faulty and unaltered information
    - E.g., DFA, DFIA.

Fault Attack Countermeasures

- Fault attack countermeasures attempt to prevent an attacker from observing the effect of injected faults.
  - Shielding to physically block fault injection
  - Sensors to detect fault injections
    - E.g., temperature, EM, voltage anomaly sensors.
  - Redundancy for fault detection
    - E.g., error-correcting codes (hardware), multi-versioning (software).

Motivation

Type	Overhead	Cost	Flexibility	Efficacy
Hardware	High	High	Low	High
Software	High	Moderate	High	Low

Table 1: Existing countermeasures on a yardstick.

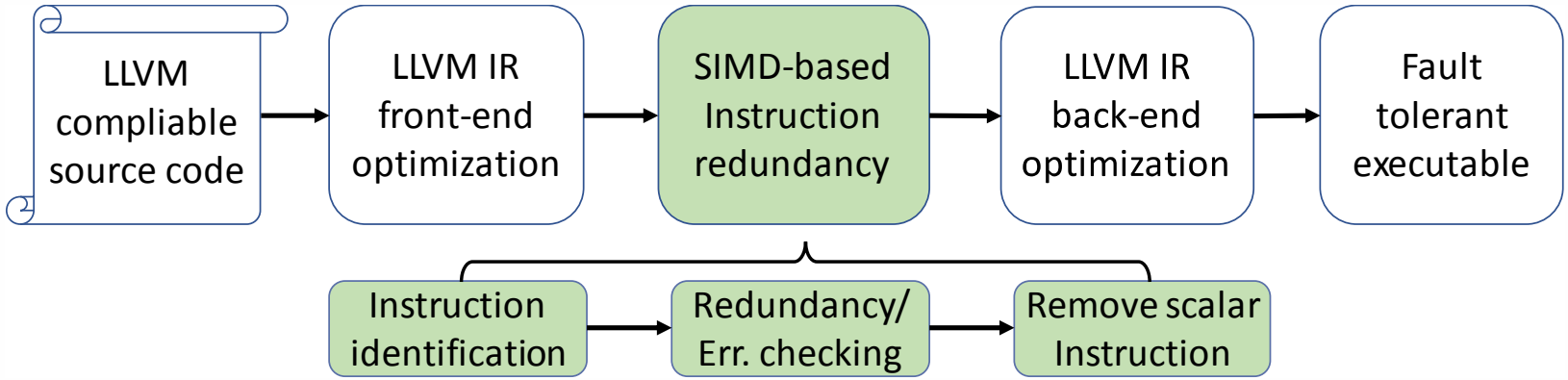
Recent research has shown that **multiple** fault injections can break redundancy techniques in algorithm or instruction level - refer to Yuce et al. in FDTC'16

- Software countermeasures (timing and spatial code redundancy)
  - Overhead impact memory footprint, code size, and register pressure, i.e., performance
    - E.g., run crypto algorithm twice, duplicate instructions
  - Tedious and error-prone deployment of the countermeasures.
  - Highly flexible.

CAMFAS

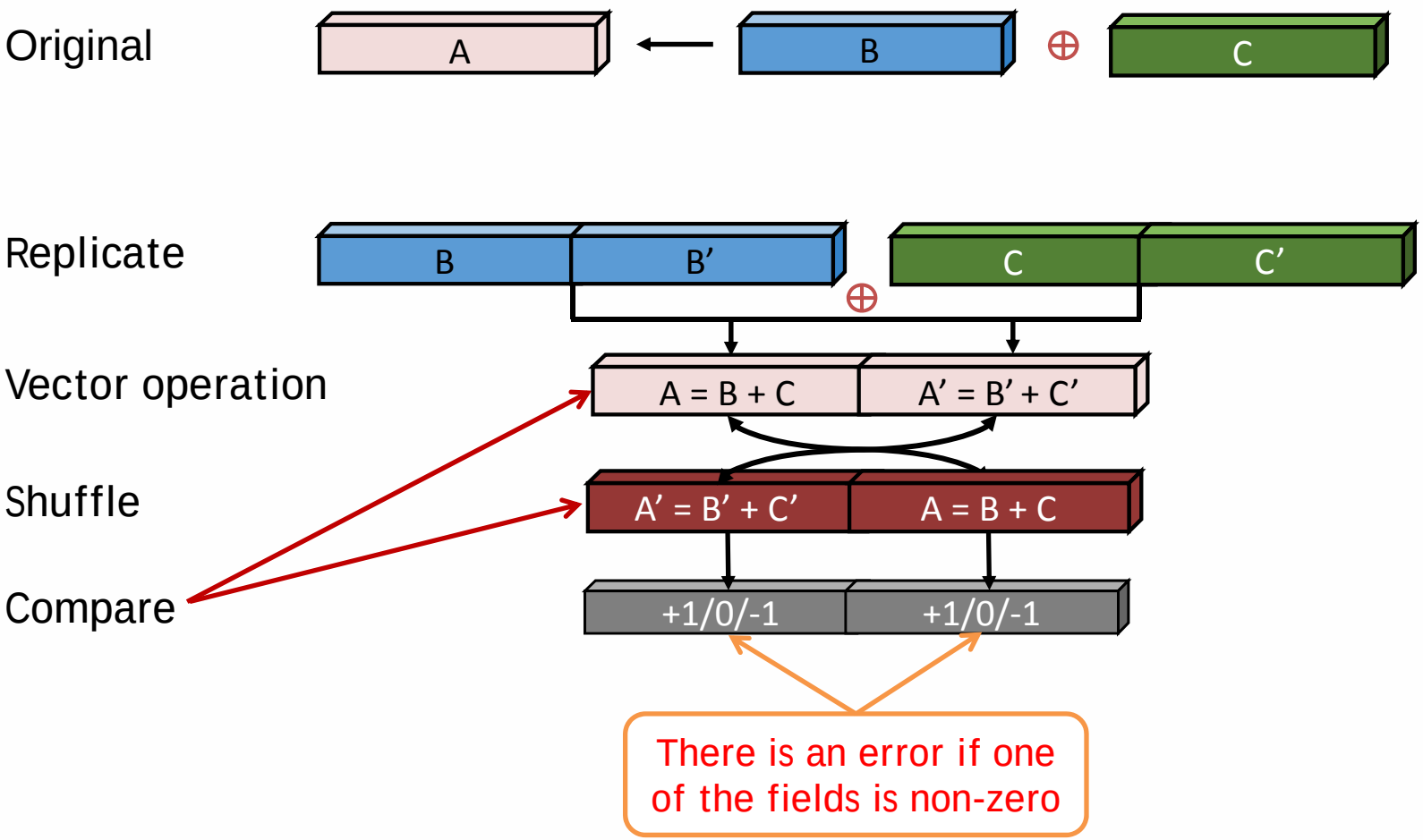
- Rationale
  - Rely on the mechanism of automatic vectorization (SIMDization) to convert instruction duplication into vector operations.
  - Vector units are ubiquitous in modern micro-processors
    - *Intel x86 – SSE, AVX*
    - *ARM - NEON*
- Expected outcome
  - Reduced performance penalty compared to instruction duplication.
  - Elevated fault coverage due to a reliable insertion of the mitigation with an enhanced version of the auto-vectorizer of the compiler: CAMFAS.

CAMFAS Framework



- ALU instructions: Duplicated and their data is placed in SIMD registers.
- Memory instructions: Memory addresses are duplicated using gather and scatter.
- Branches: Condition computation is duplicated. PC update is not checked.
- Calls: Function calls are not duplicated, but some library calls are duplicated if they have the equivalent SIMD prototypes in LLVM IR (e.g. sqrt, pow, etc.).

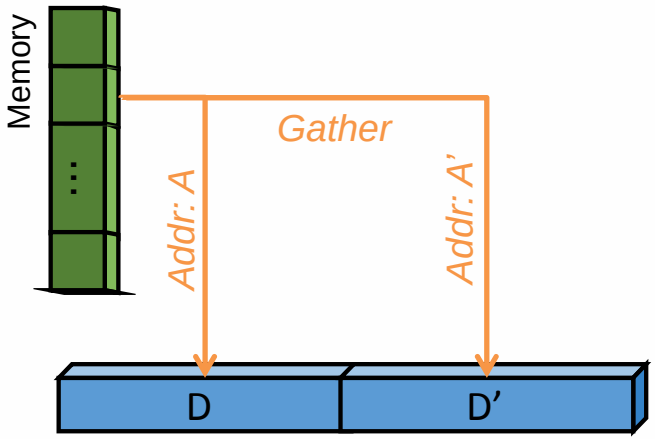
ALU Instruction Duplication



Memory Instruction Duplication

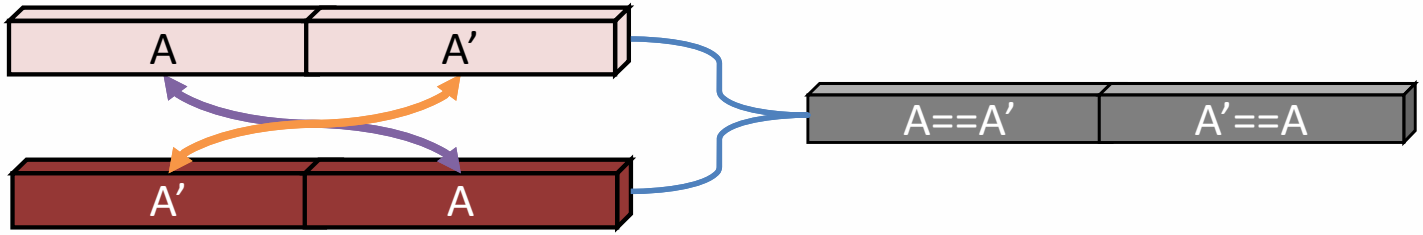
- Only addresses are protected.
  - Load instruction: gather.
  - Store instruction: address is checked before store.
- Data can also be checked at the cost of more overhead

Original: D = Mem[A]



Error Checking Insertion

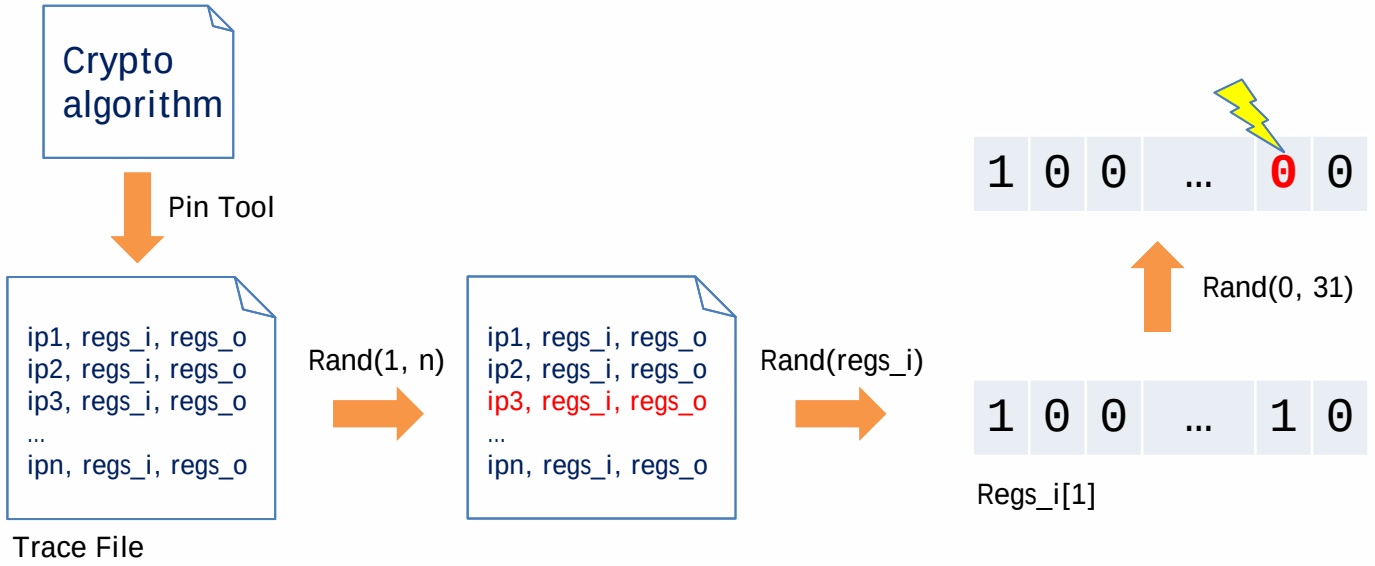
- Shuffle + Compare.



- Error checking code only inserted at *selected* positions to reduce the performance overhead.
  - Before stores.
  - Before function calls.
  - Before conditional branches.

Fault Injection Simulation

- Use Pin tool to collect dynamic instruction trace.
- Randomly pick a fault position in trace file.
  - Pick an instruction → pick a register → pick a bit
  - Flip the chosen bit
- Repeat fault injection 1000 times for each crypto algorithm.



Evaluation

Experimental Platform	
CPU	Intel Xeon Phi™ 7210 with AVX-512 SIMD extension
Memory	16GB
OS	Ubuntu Server 16.04 with Linux-4.4.0 kernel
Compiler framework	LLVM 4.0
Target	Libgcrypt-1.7.6

- CAMFAS can also be applied to other micro-processors support vector extensions.
  - e.g. ARM processors w/ NEON

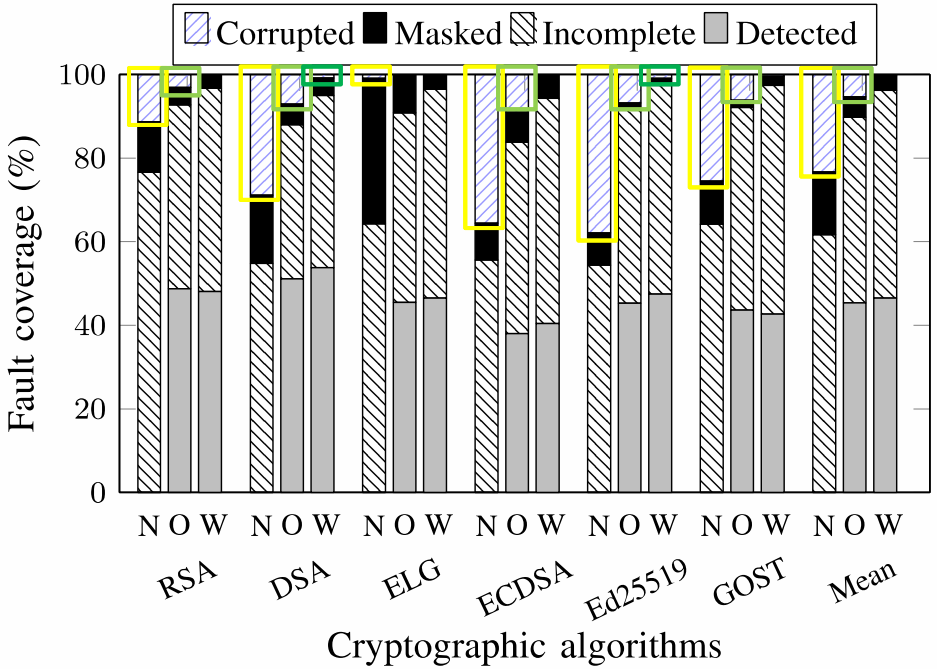
Cipher Execution Results

- *Detected*: program terminates due to fault being detected.
- *Incomplete*: execution fails *without* generating attackable output.
- *Masked*: program completes normally and produces *correct* output.

- ***Corrupted***: program completes normally and produces *faulty* output.

The fault provides useful information to an attacker for its fault analysis step

Fault Coverage



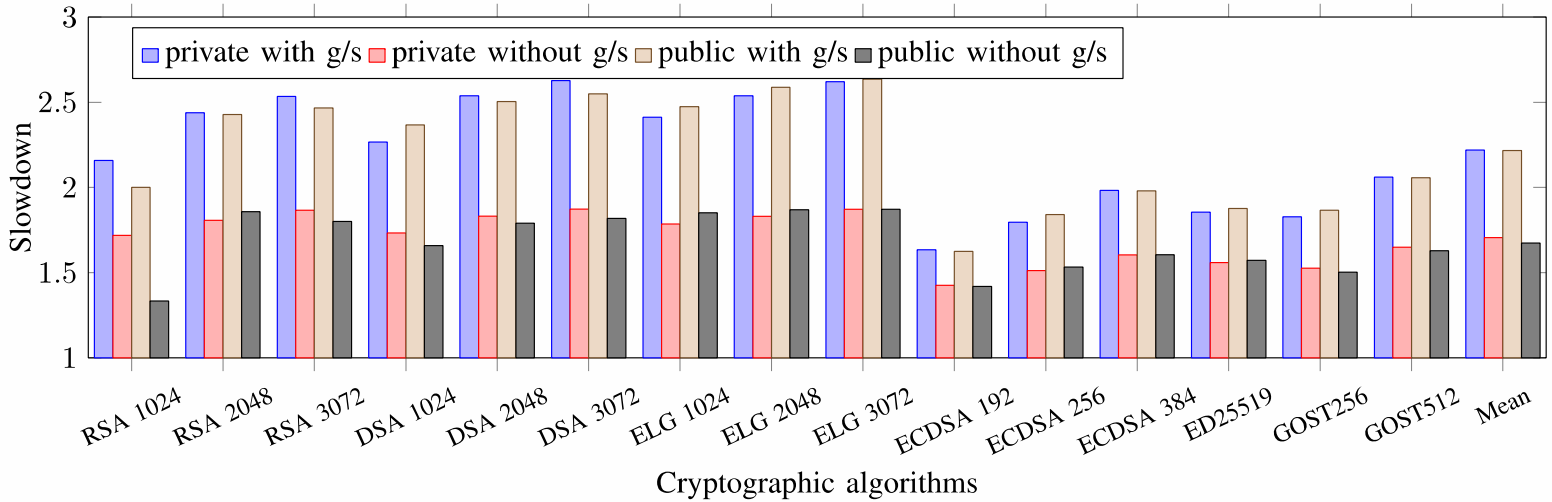
Approach	Detected	Incomplete	Masked	Corrupted
N	0	61.65%	15%	23.35%
O	45.37%	44.43%	5.83%	5.37%
W	46.5%	49.72%	3.42%	0.36%

Almost full coverage with memory protection!

- The remaining corrupted cases are mainly caused by faults injected to error checking code

N: No fault detection
O: CAMFAS without memory protection
W: CAMFAS with memory protection

Performance overhead



With memory protection: 2.2x.
Without memory protection: 1.7x.

Discussion

- Differential Fault Analysis (DFA)
  - Requires both correct and faulty cipher texts.
  - CAMFAS detects incorrect result and prevents the generation of faulty cipher text.
- Differential Fault Intensity Analysis (DFIA)
  - Relies on the bias of fault behavior.
  - CAMFAS effectively prevents faulty output from being propagated.
- Single-Glitch Attack
  - Injects clock glitches at precisely controlled timing and pipeline stages to thwart redundancy-based countermeasures
  - CAMFAS makes the attack more difficult as duplication and error checking are inserted at the IR level

Conclusions

- CAMFAS is a redundancy-based countermeasure implemented in LLVM infrastructure.
- CAMFAS exploits SIMD units in modern micro-processors to mitigate fault attacks.
- CAMFAS provides high fault coverage while keeps a moderate performance penalty.

Future directions

- Extend CAMFAS to thwart side-channel attacks.
- New micro-architectural features to mitigate fault attacks.

Thank you!