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)

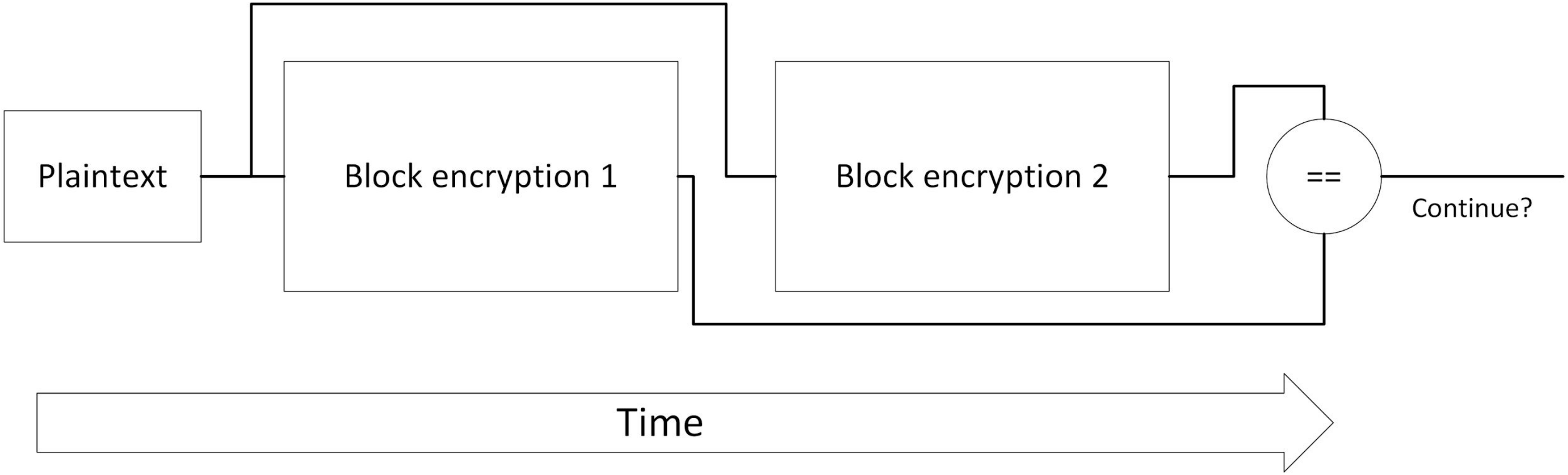
Fault attack countermeasures

- 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

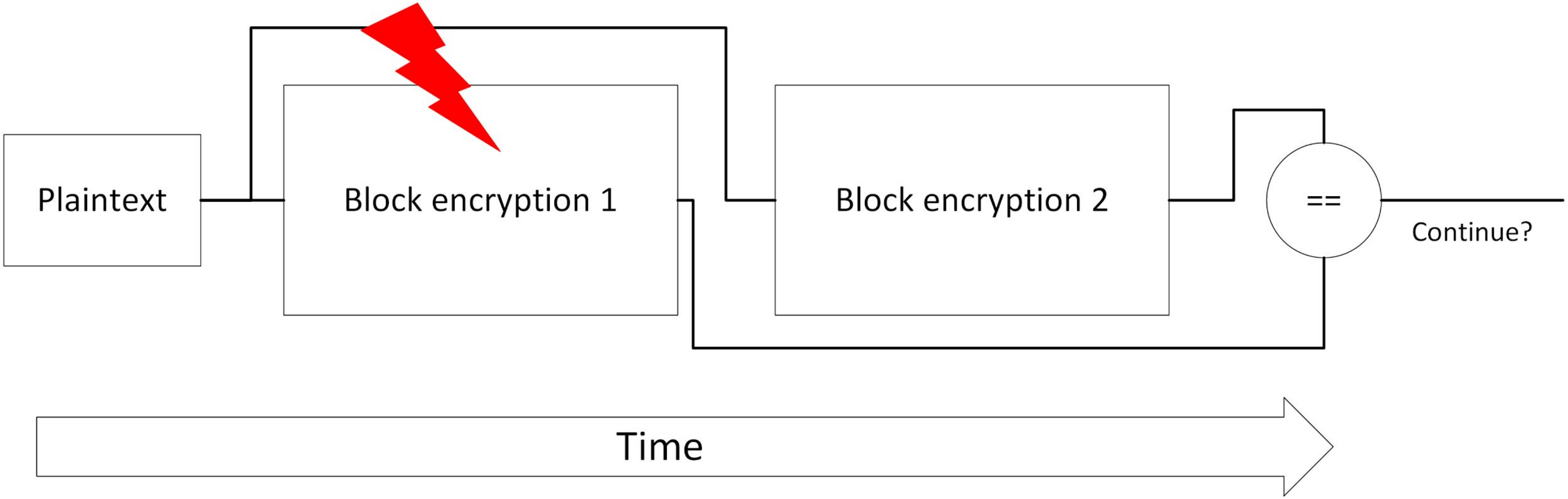
Our motivation

- Hardware solutions are expensive and slow to market
- Can we resist fault attacks using only software?

Software countermeasures: Algorithm Duplication

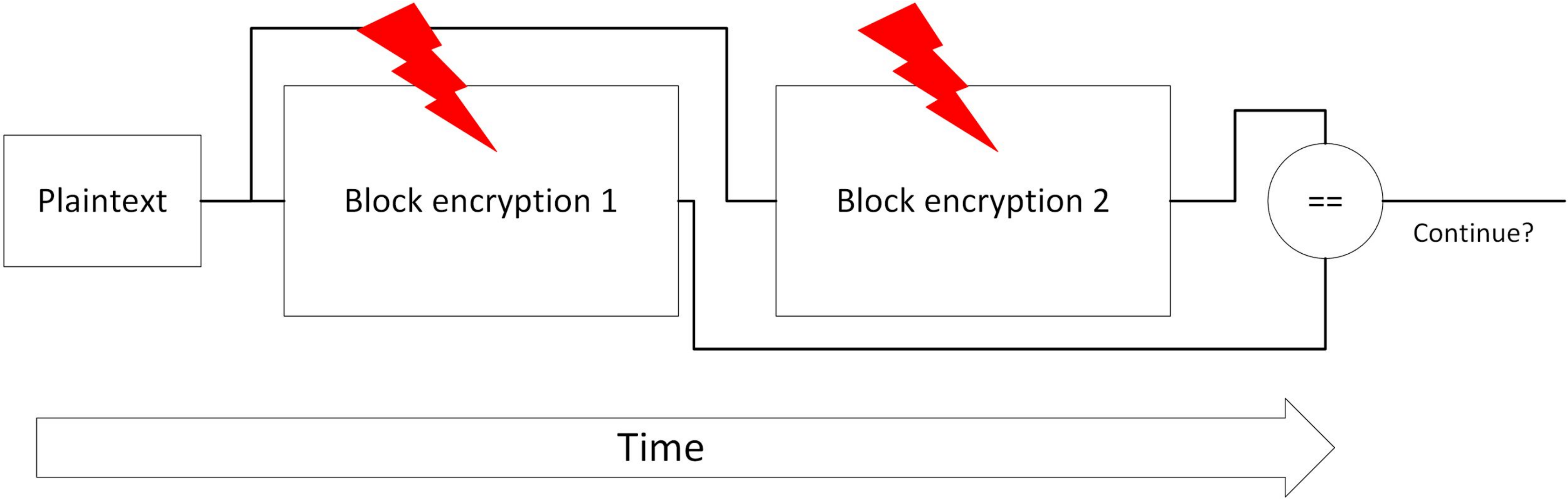


Software countermeasures: Algorithm Duplication



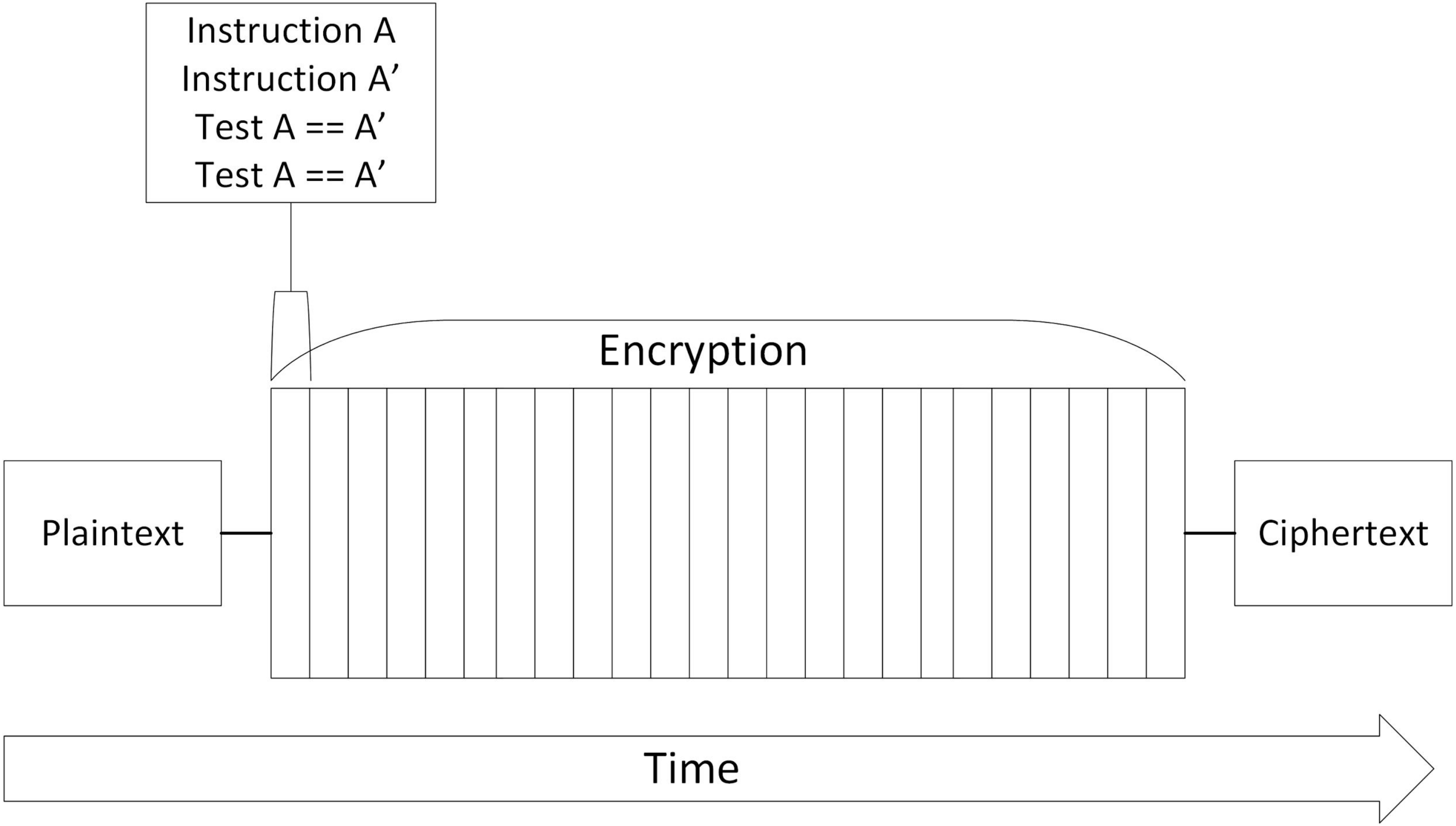
Detected!

Software countermeasures: Algorithm Duplication

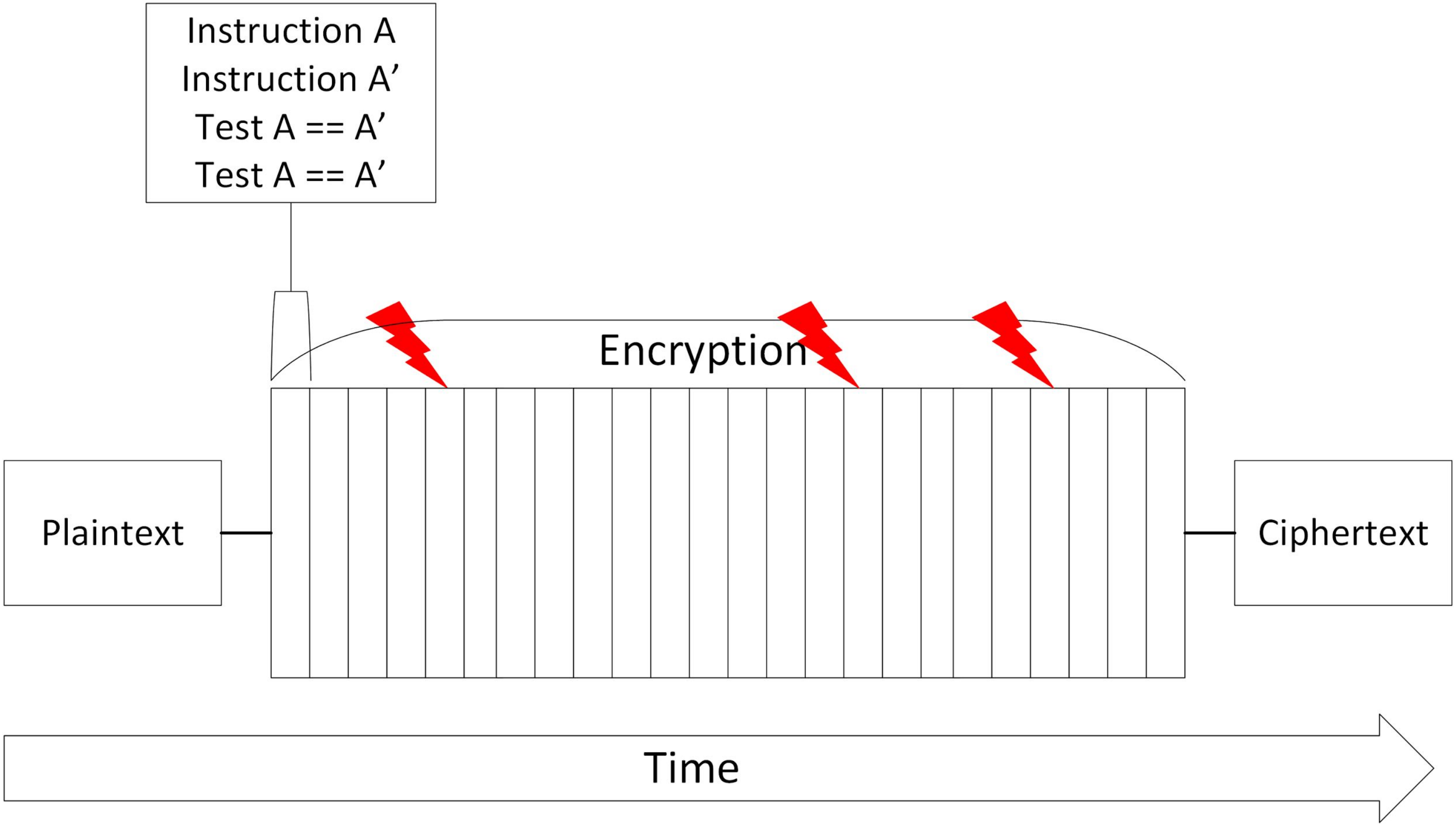


Not detected!

Software countermeasures: Instruction Duplication

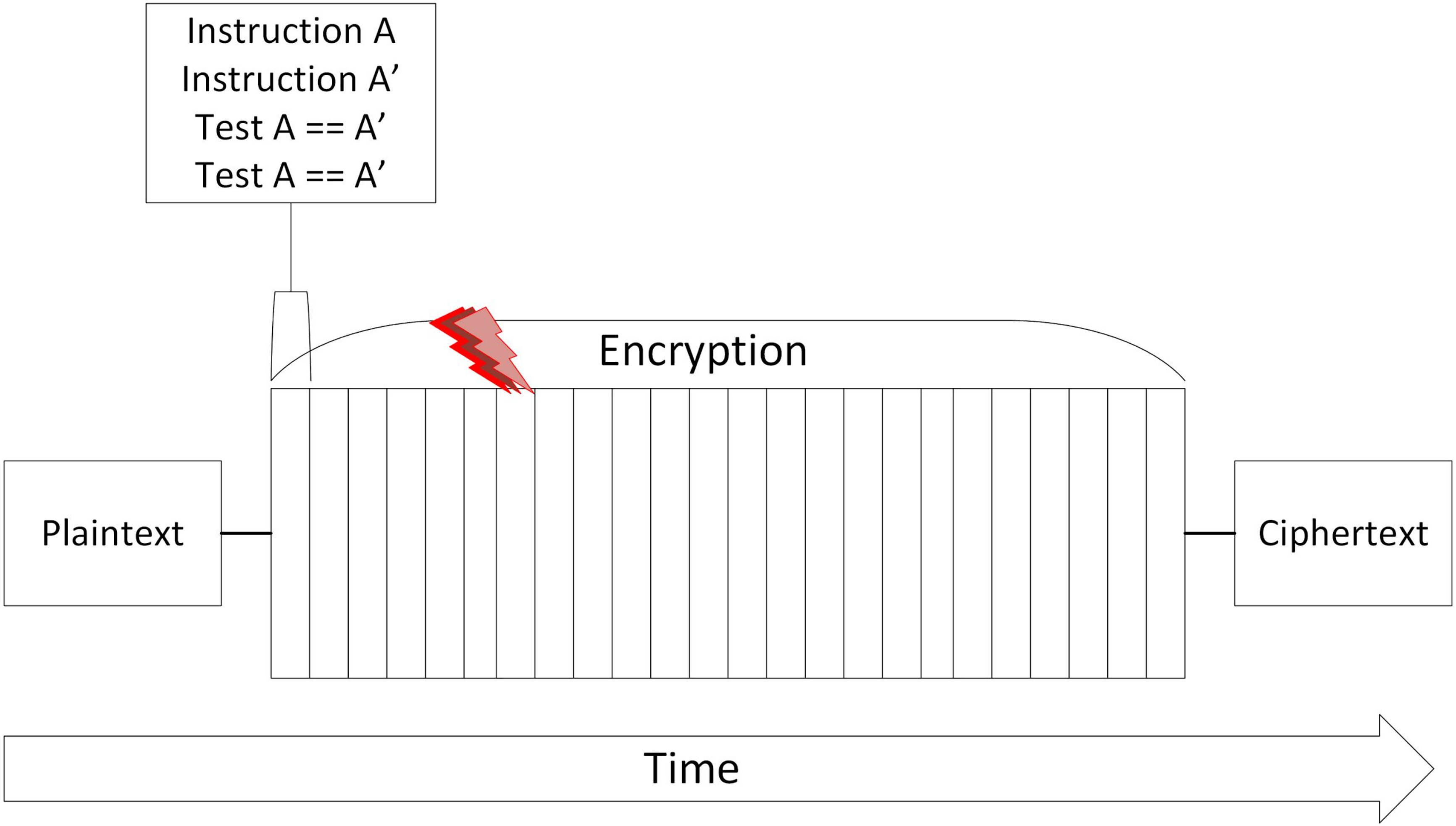


Software countermeasures: Instruction Duplication



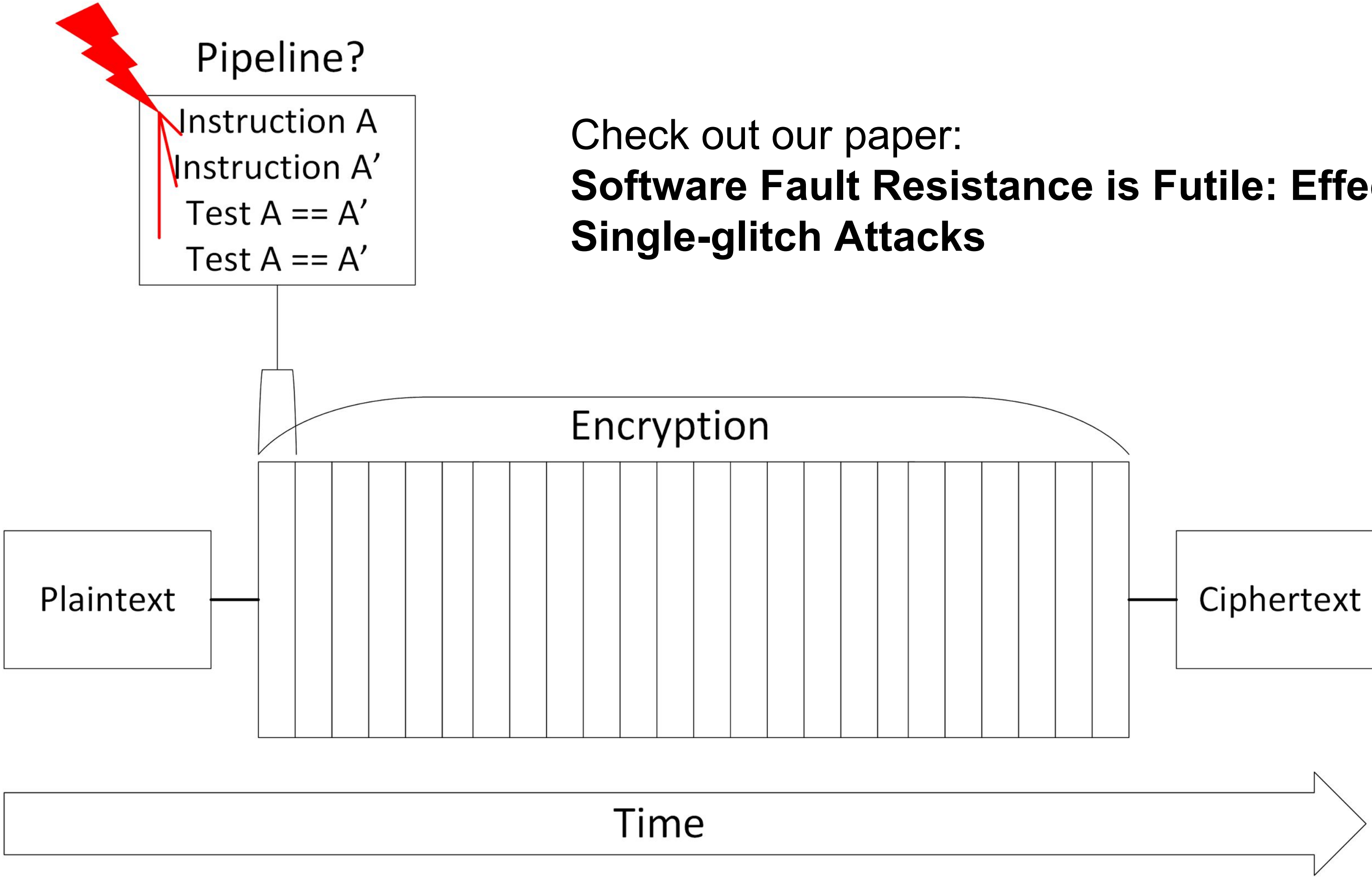
Detected!

Software countermeasures: Instruction Duplication



Not detected!

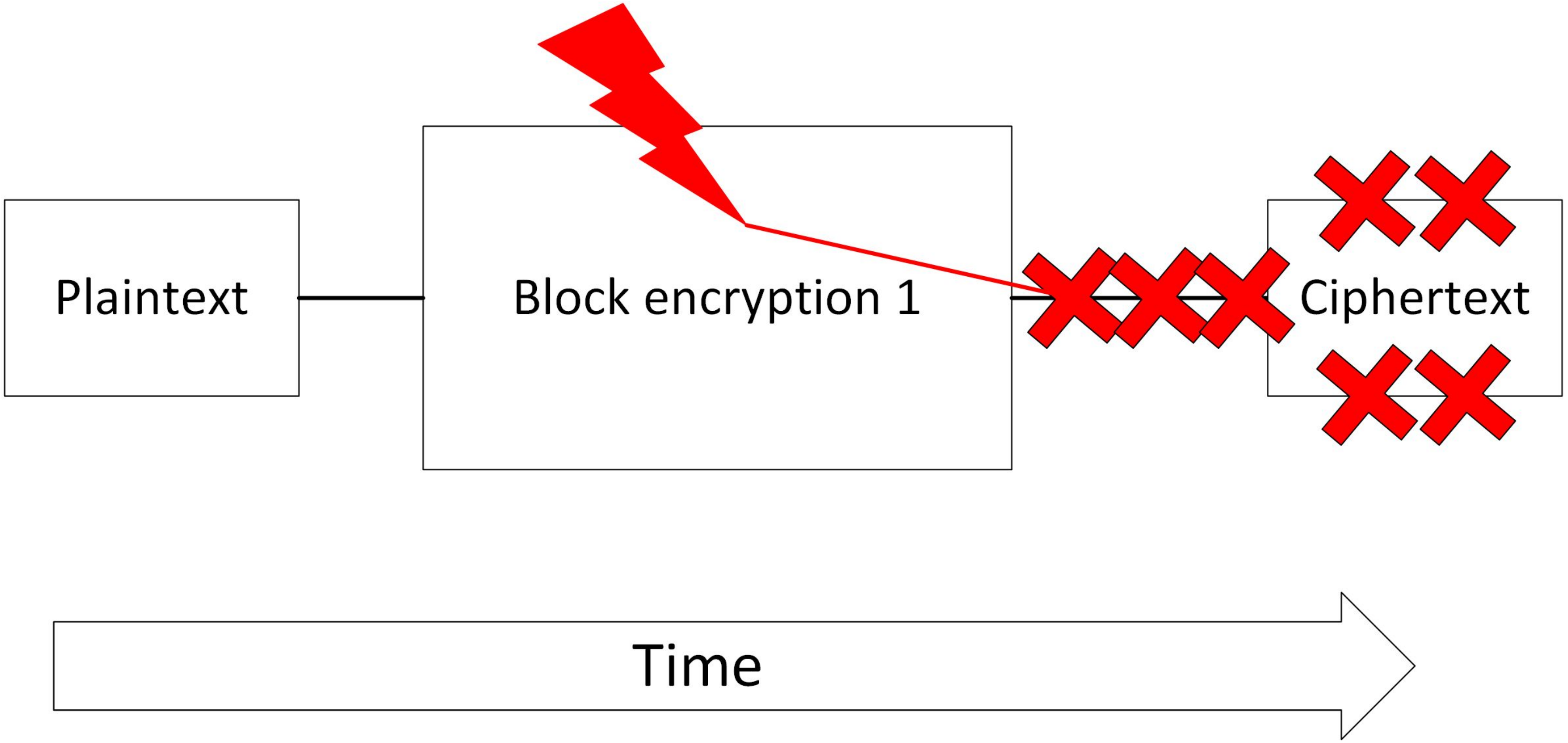
Software countermeasures: Instruction Duplication



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

Not Detected!

Software countermeasures: Infective

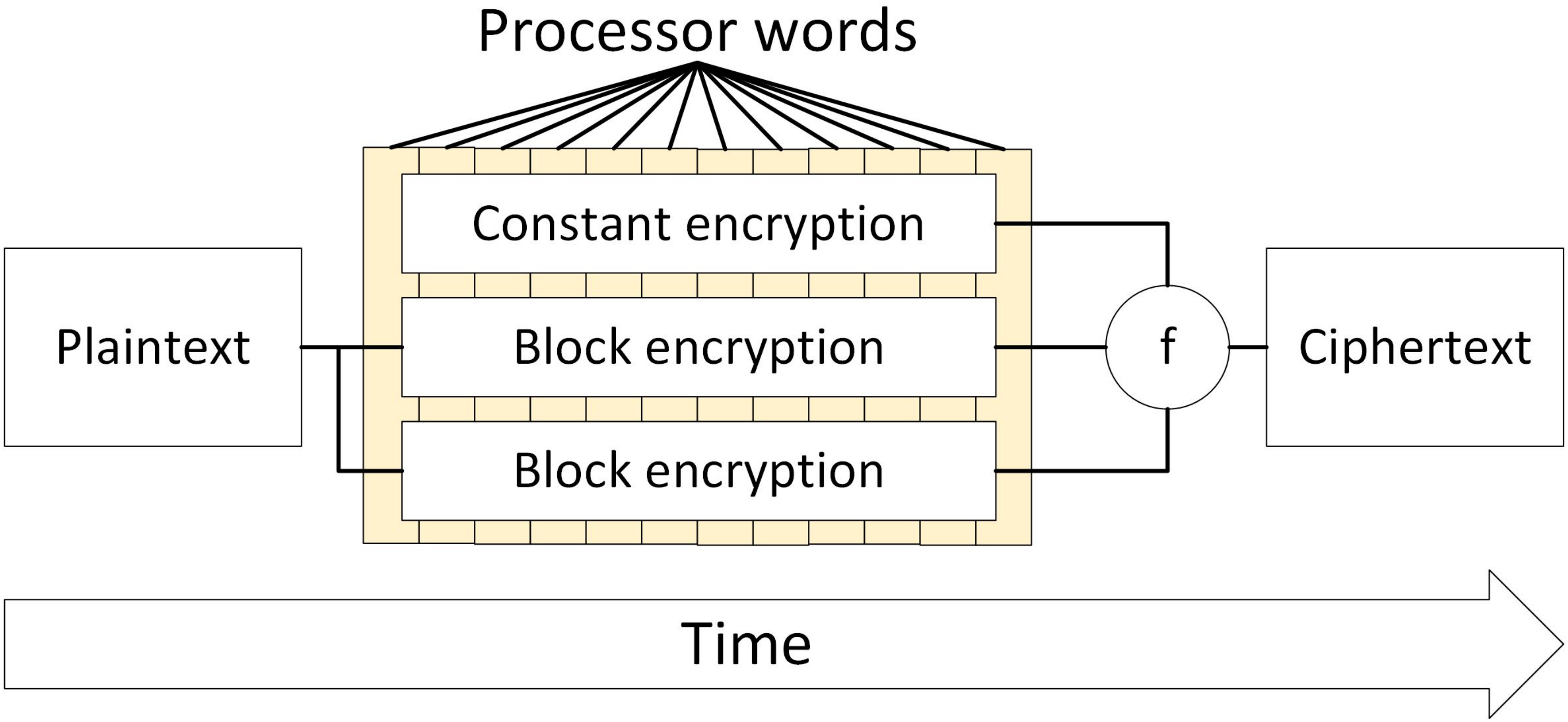


All attempts have been broken

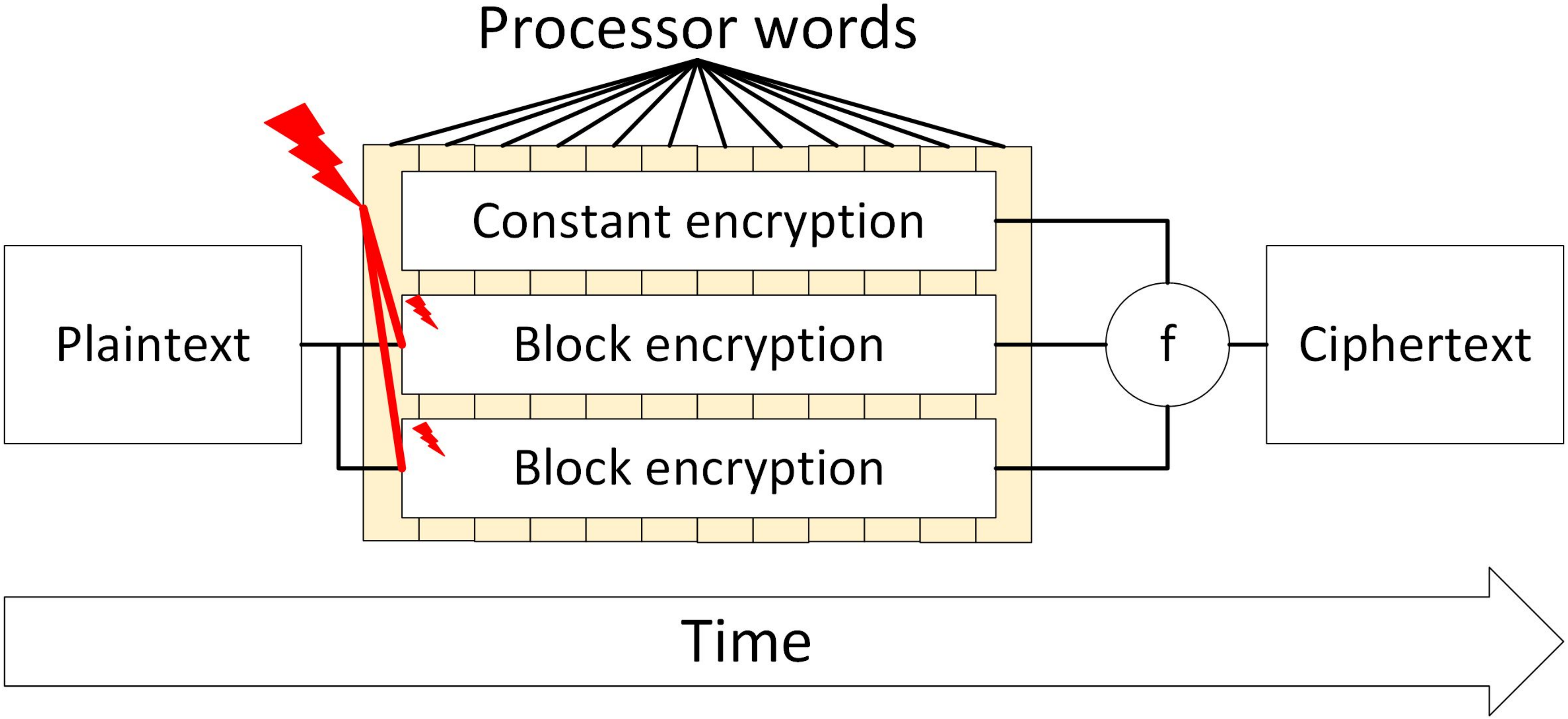
Intra-Instruction Redundancy (IIR)

- 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)



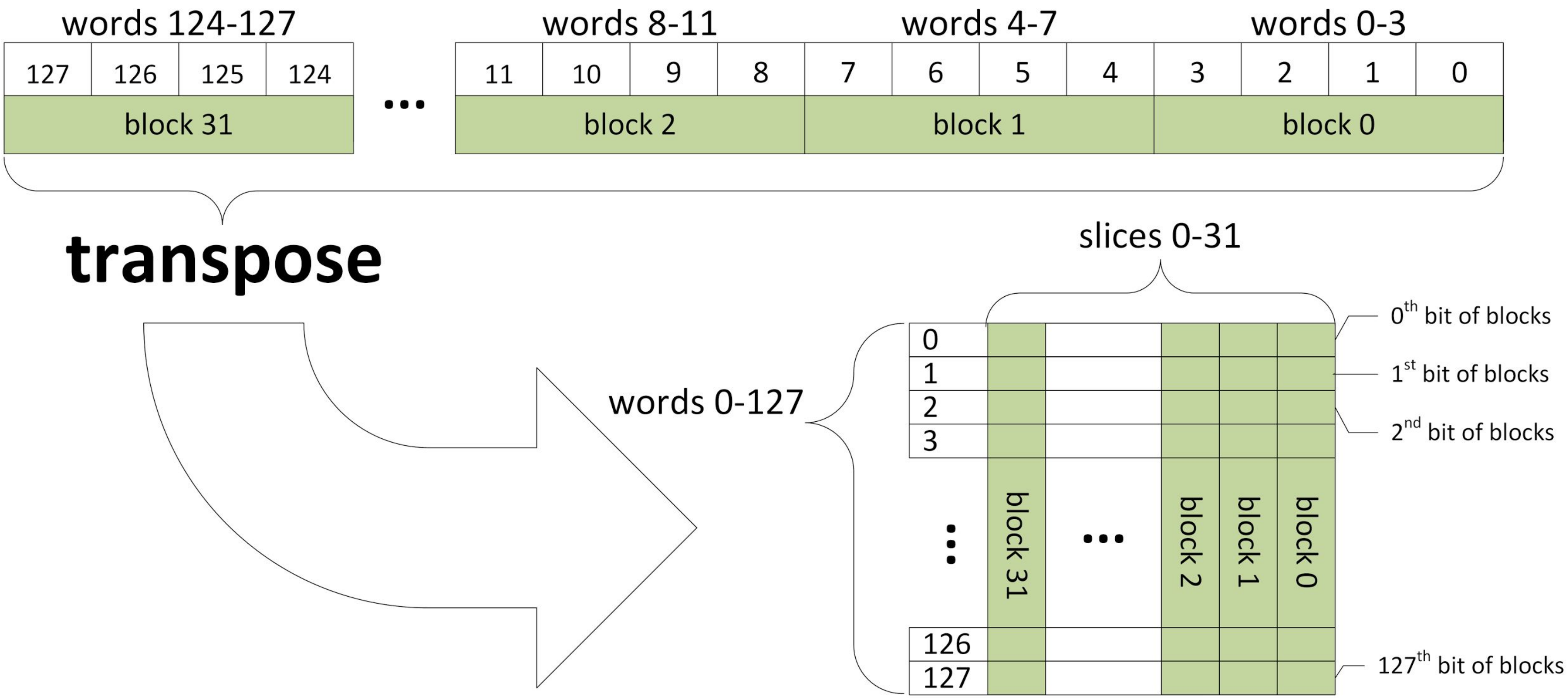
Our software countermeasure: Intra-Instruction Redundancy (IIR)



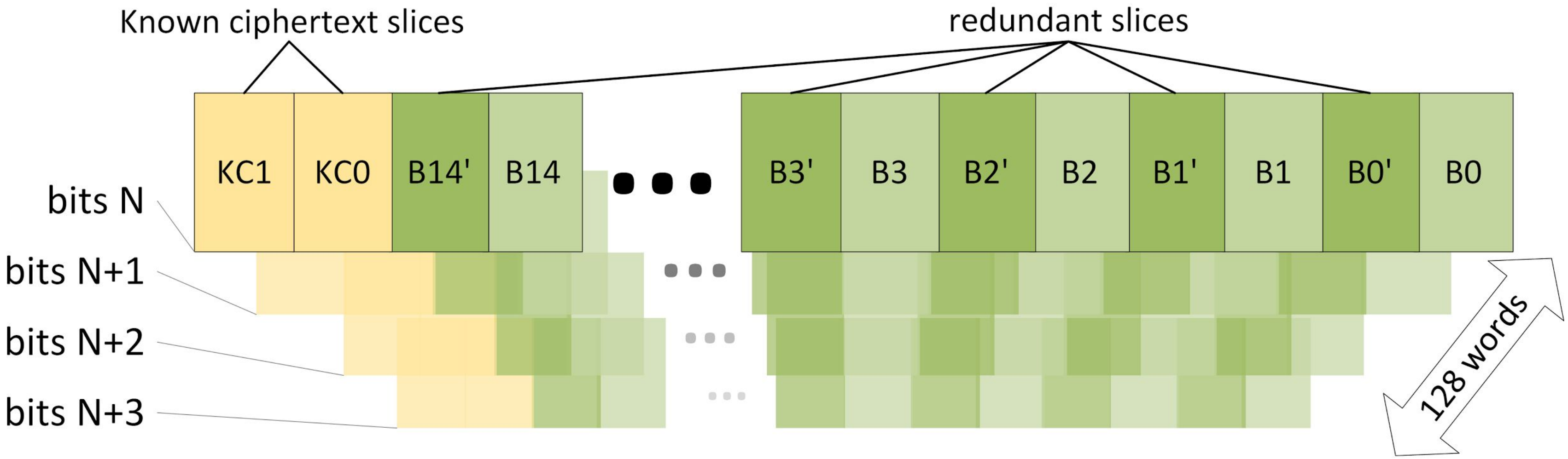
An adversary must make a target 2 bit fault in a processor word

How to implement? With bit-slicing.

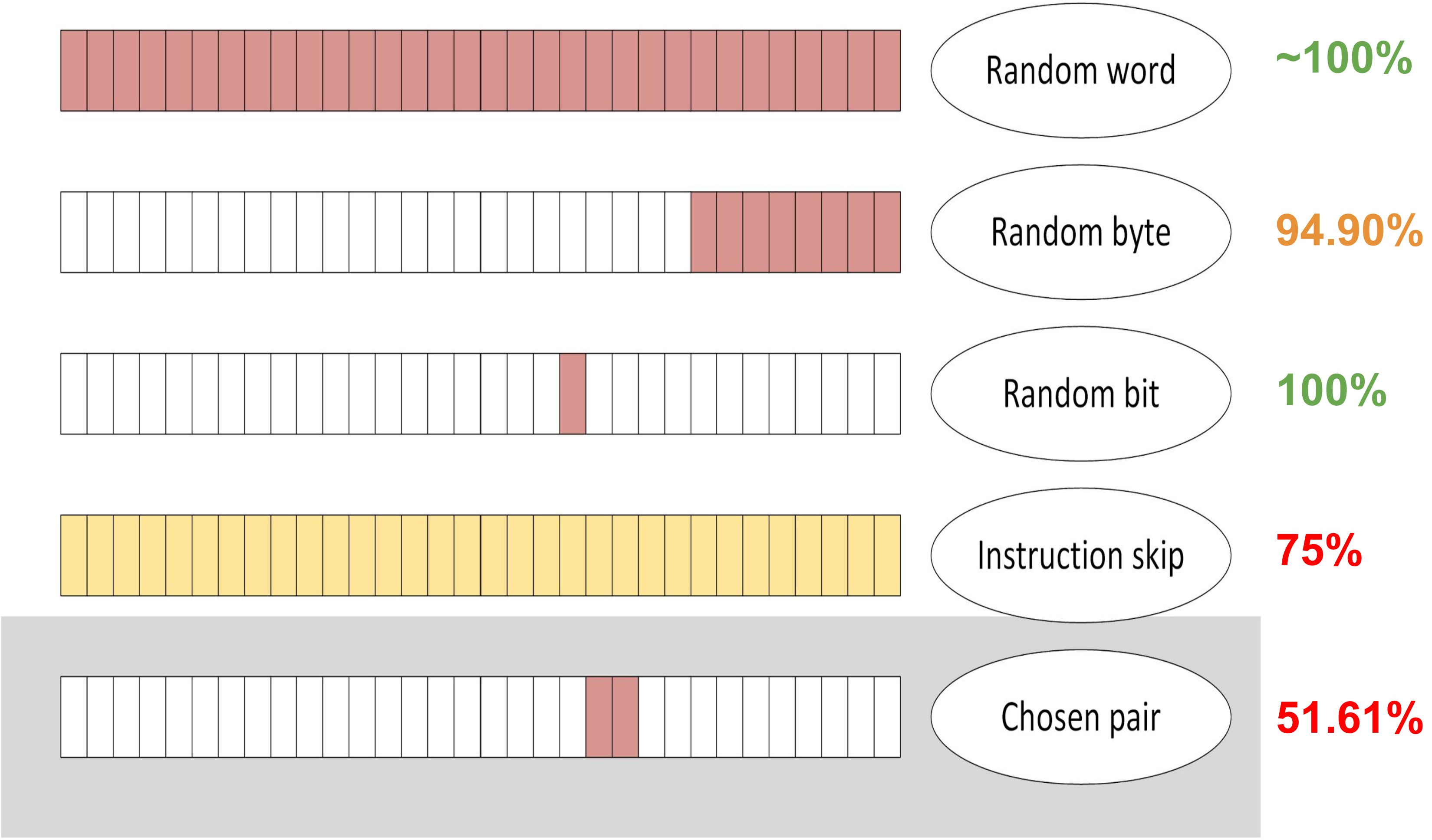
- 32 bit processor word
- 32, 128-bit blocks to encrypt



IIR Slice Allocation



Theoretical Fault Coverage



Random word ~100%

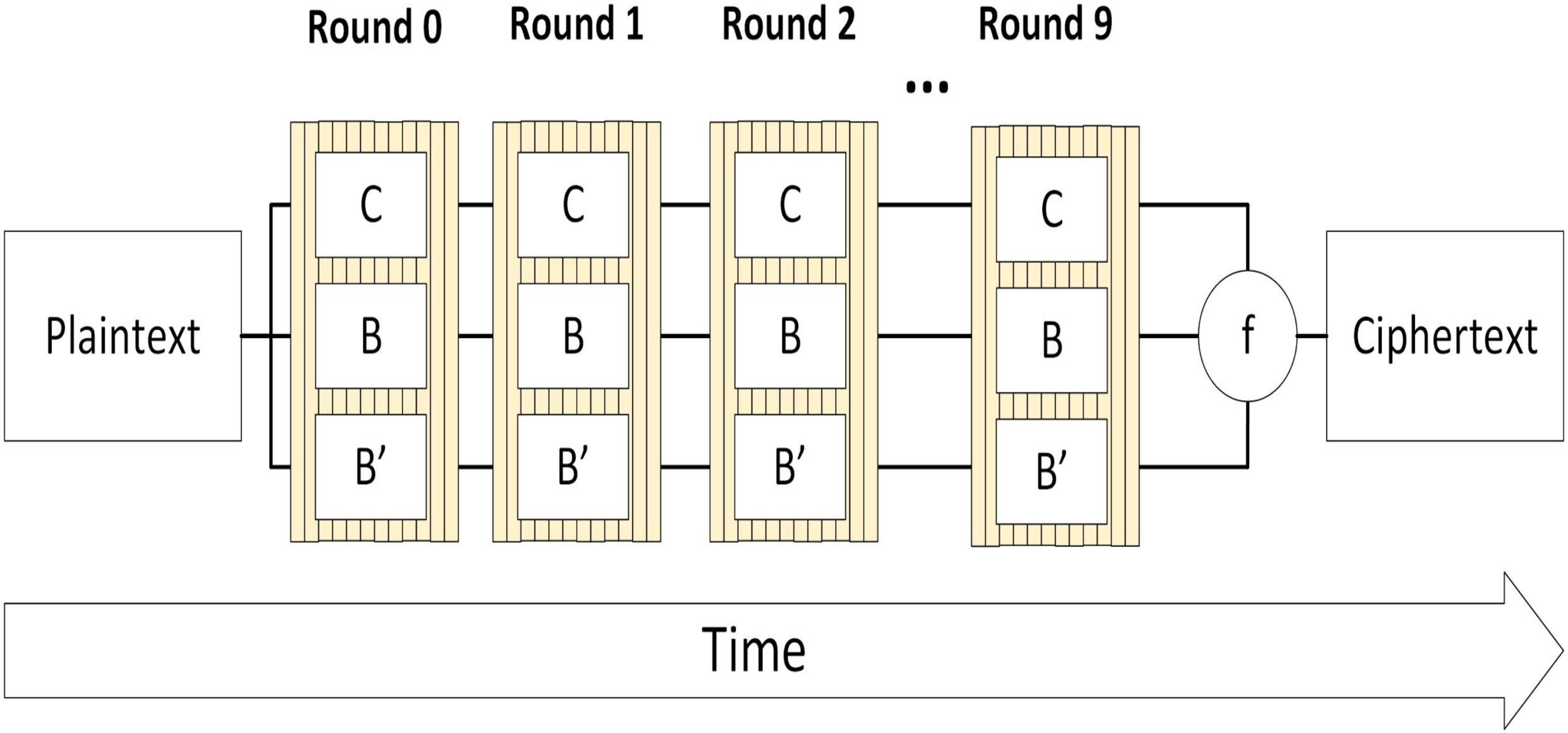
Random byte 94.90%

Random bit 100%

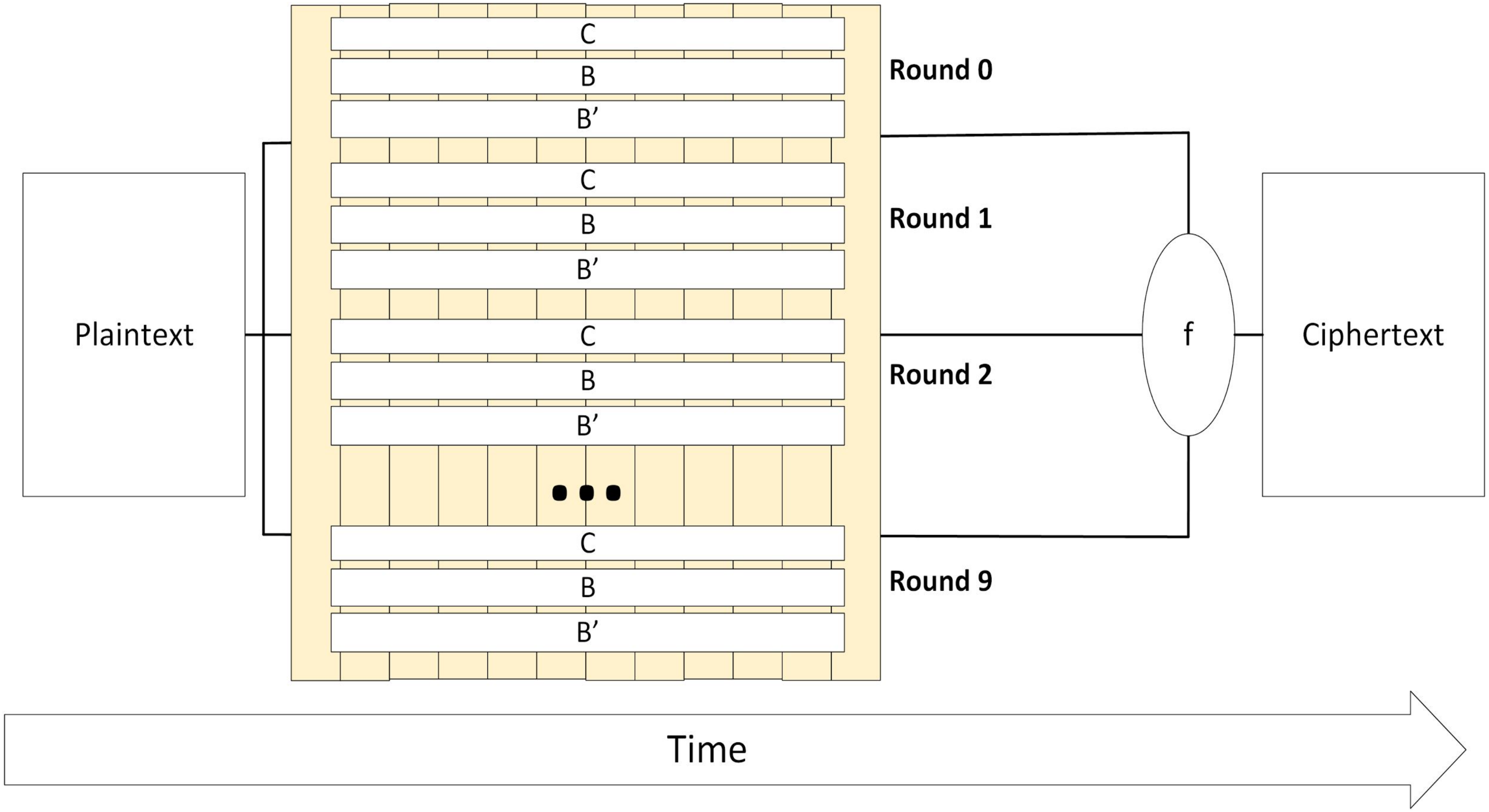
Instruction skip 75%

Chosen pair 51.61%

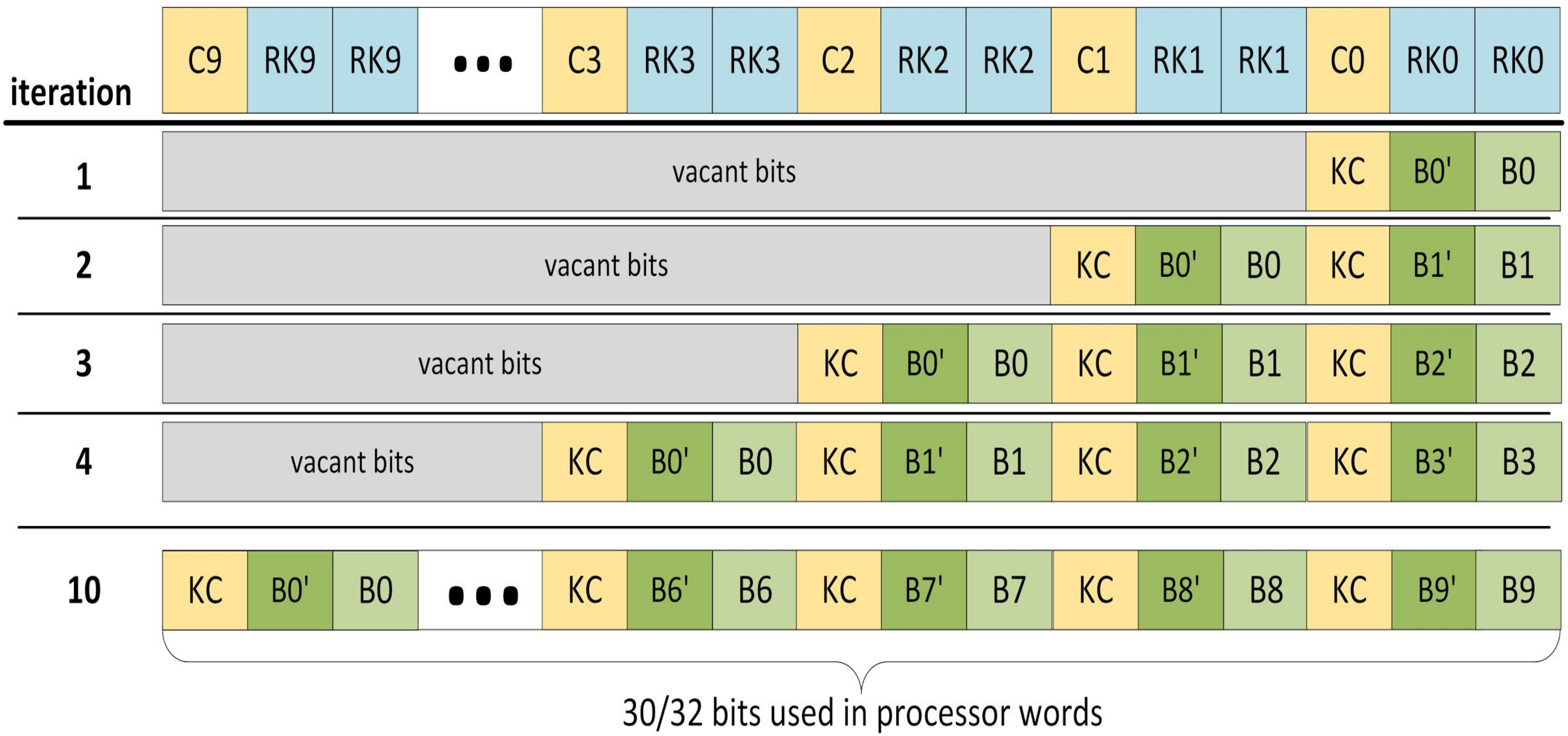
Problem: rounds are time separated



Solution: make each slice a different round



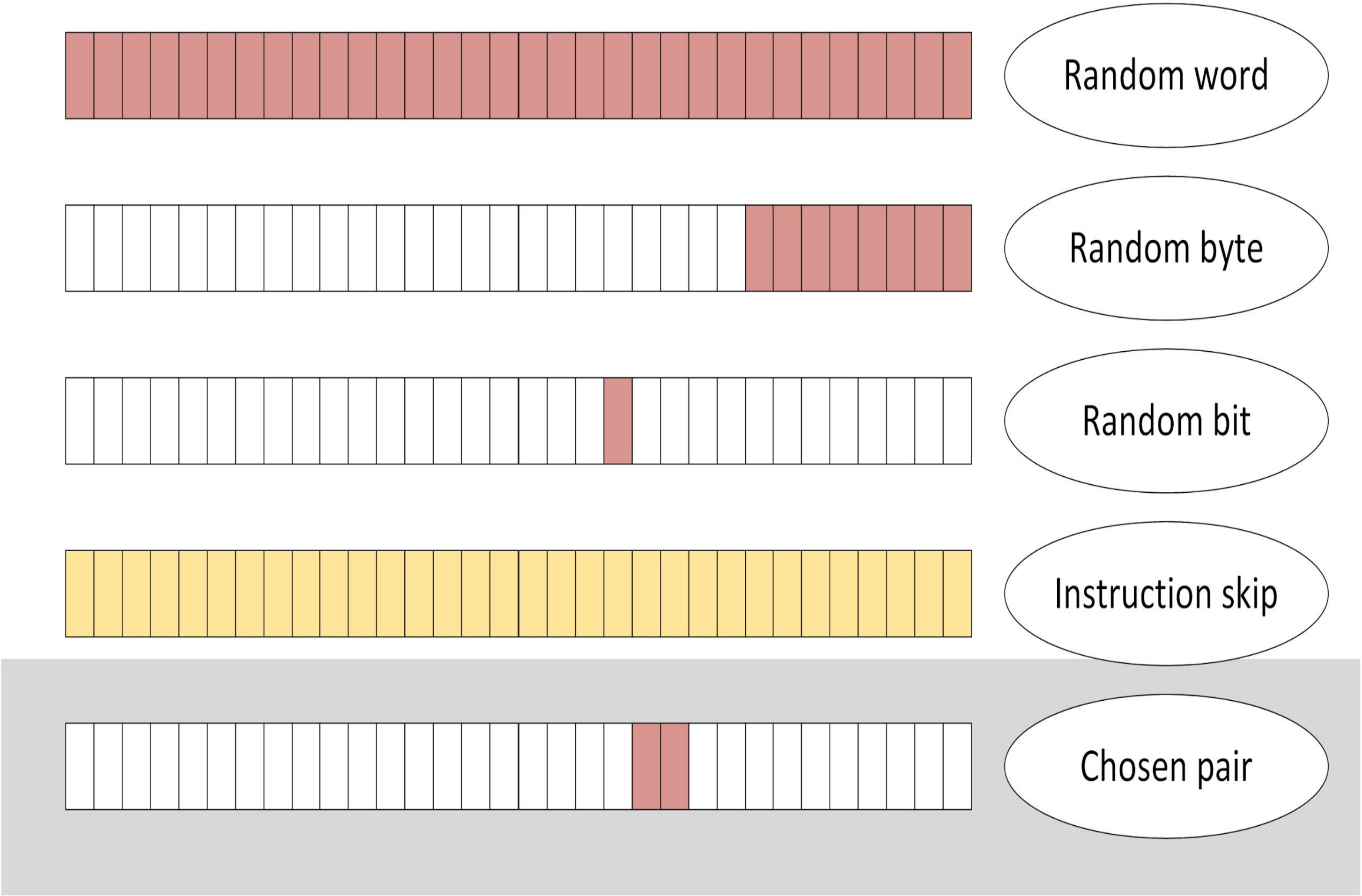
Improving IIR by adding Pipelining



iteration	C9	RK9	RK9	•••	C3	RK3	RK3	C2	RK2	RK2	C1	RK1	RK1	C0	RK0	RK0
1	vacant bits													KC	B0'	B0
2	vacant bits										KC	B0'	B0	KC	B1'	B1
3	vacant bits							KC	B0'	B0	KC	B1'	B1	KC	B2'	B2
4	vacant bits				KC	B0'	B0	KC	B1'	B1	KC	B2'	B2	KC	B3'	B3
10	KC	B0'	B0	•••	KC	B6'	B6	KC	B7'	B7	KC	B8'	B8	KC	B9'	B9

30/32 bits used in processor words

Theoretical Fault Coverage



Random word ~100%

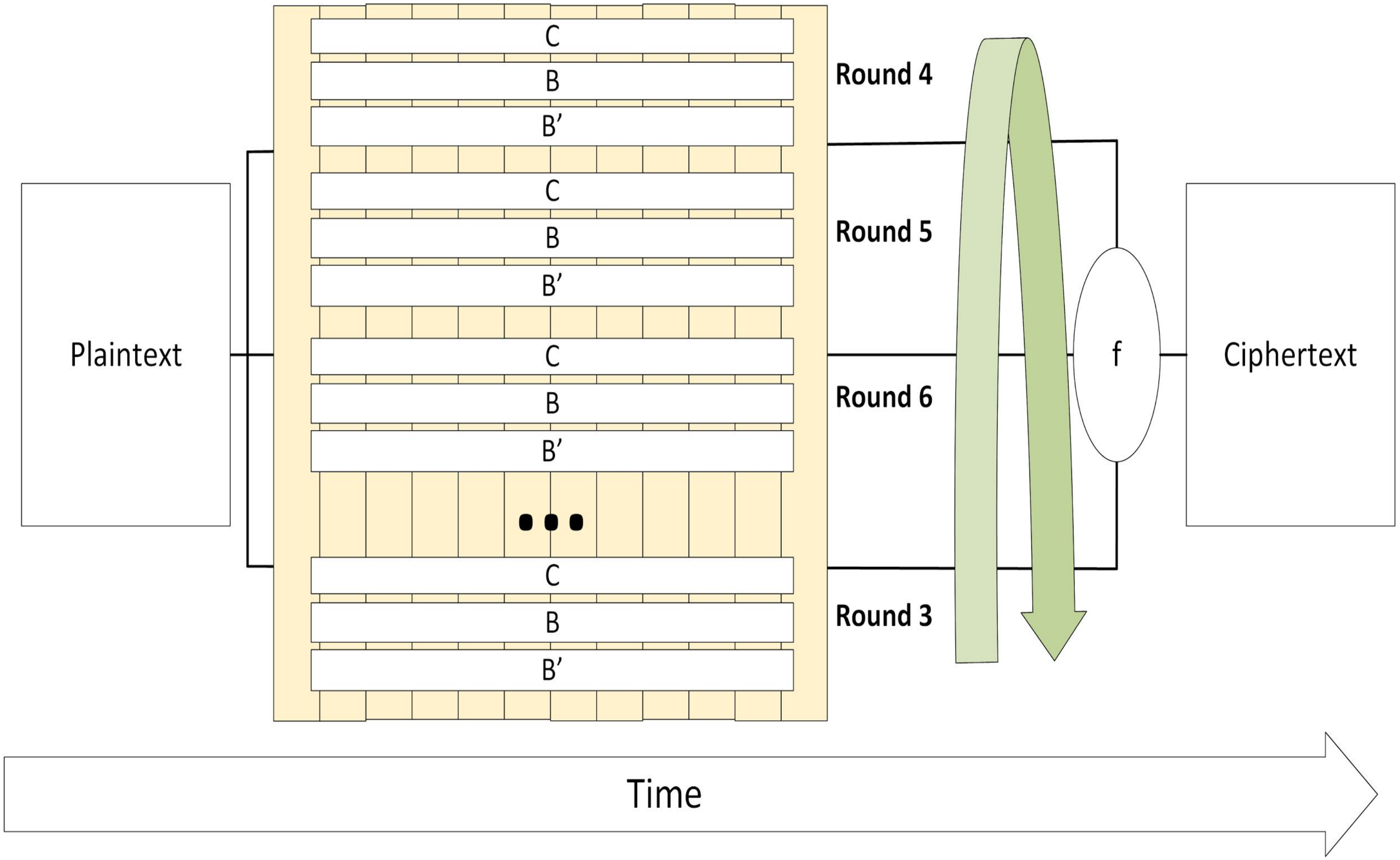
Random byte 99.90%

Random bit 100%

Instruction skip 99.90%

Chosen pair 96.77%

More: add random shifts



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:

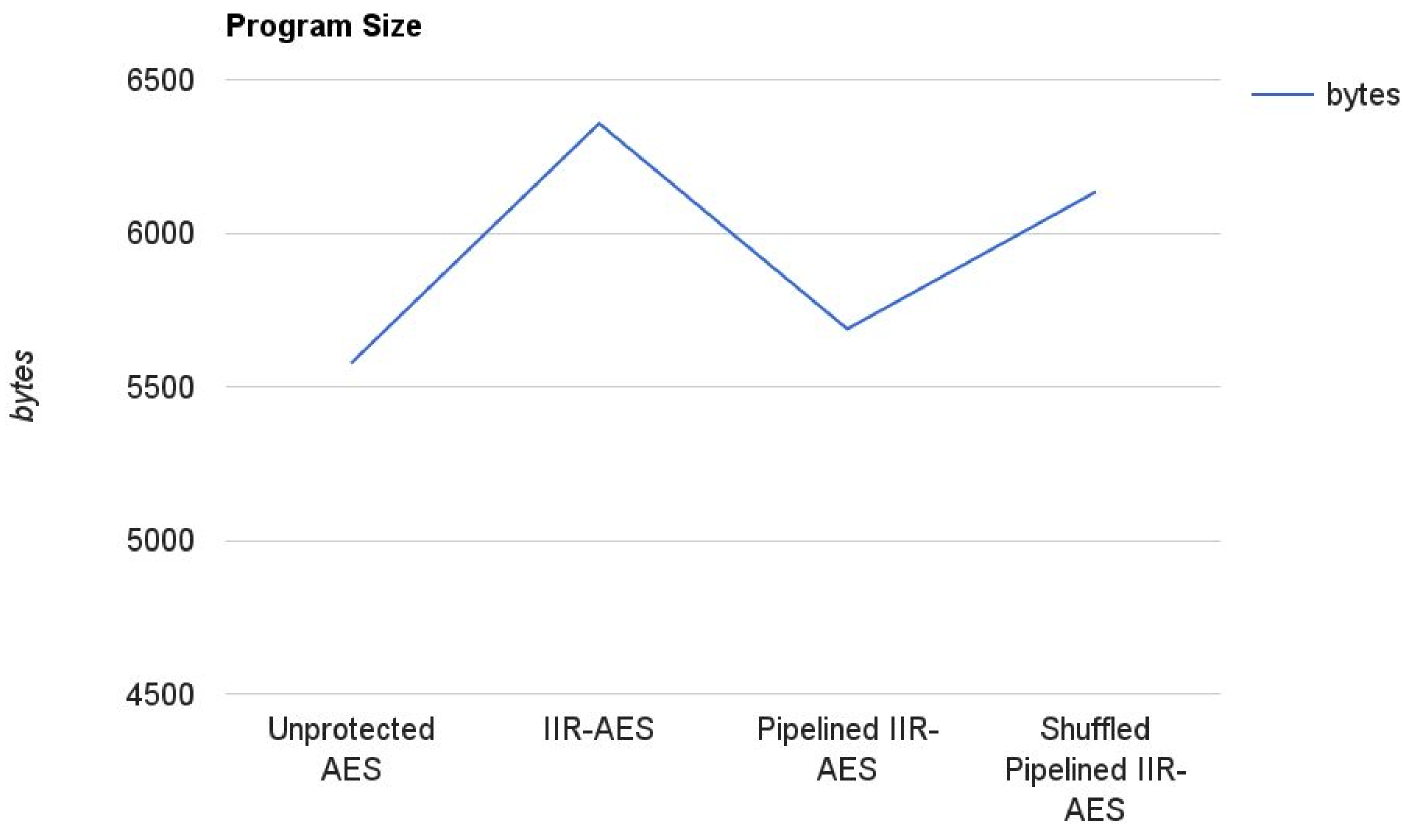
Performance	**Program size**
469.3 cycles/byte	5576 bytes

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

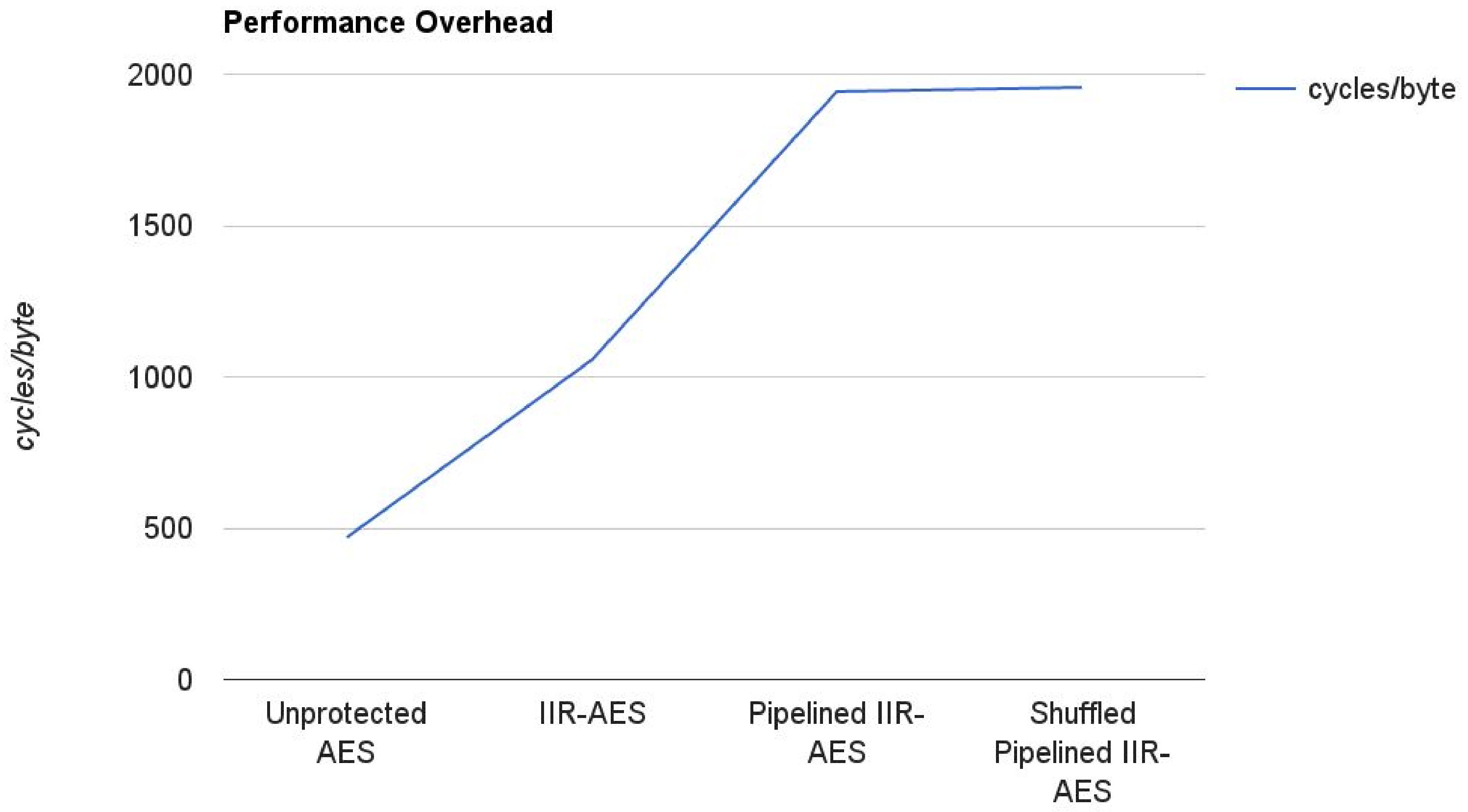
Countermeasure Overhead

	Performance	**Footprint**
Unprotected AES	469.3 cycles/byte	5576 bytes
IIR-AES	1055.9 cycles/byte	6357 bytes
Pipelined IIR-AES	1942.9 cycles/byte	5688 bytes
Shuffled Pipelined IIR-AES	1957 cycles/byte	6134 bytes

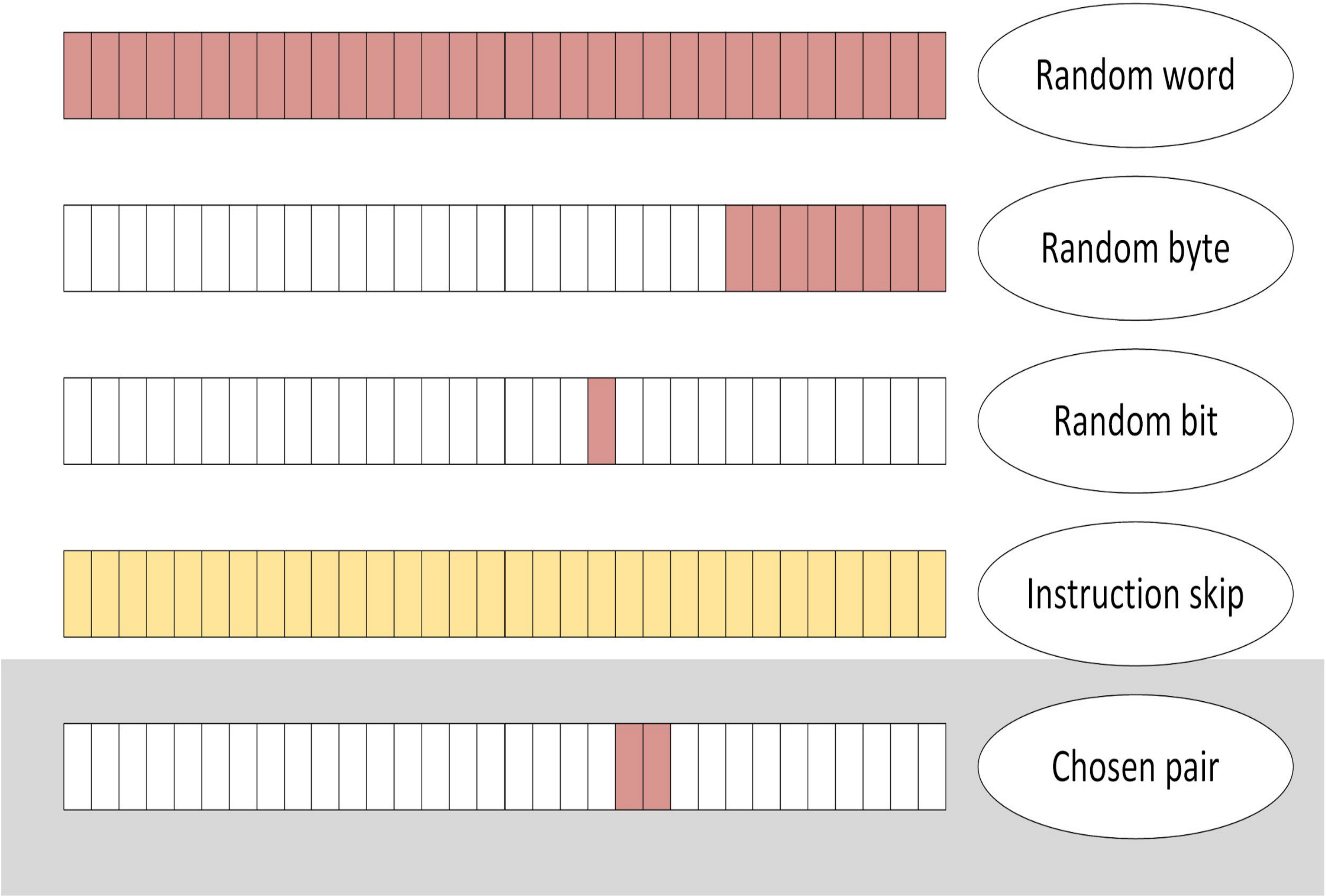
Countermeasure Program Size Overhead



Countermeasure Performance Overhead



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