Fault-Channel Watermarks

Peter Samarin[1,2], Alexander Skripnik[1], and Kerstin Lemke-Rust[1]

Bonn-Rhein-Sieg University of Applied Sciences[1]
Ruhr-Universität Bochum[2]
Germany

27 September 2016

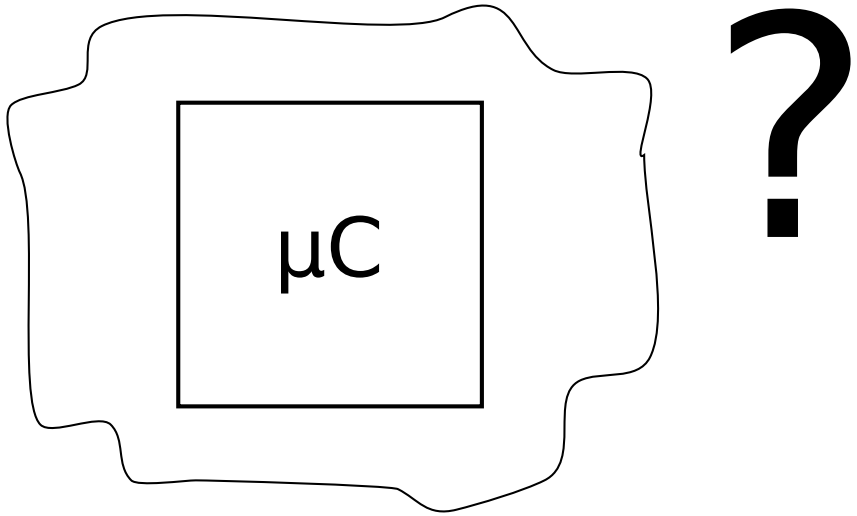
Bonn-Rhein-Sieg
University of Applied Sciences

RUHR
UNIVERSITÄT
BOCHUM

RUB

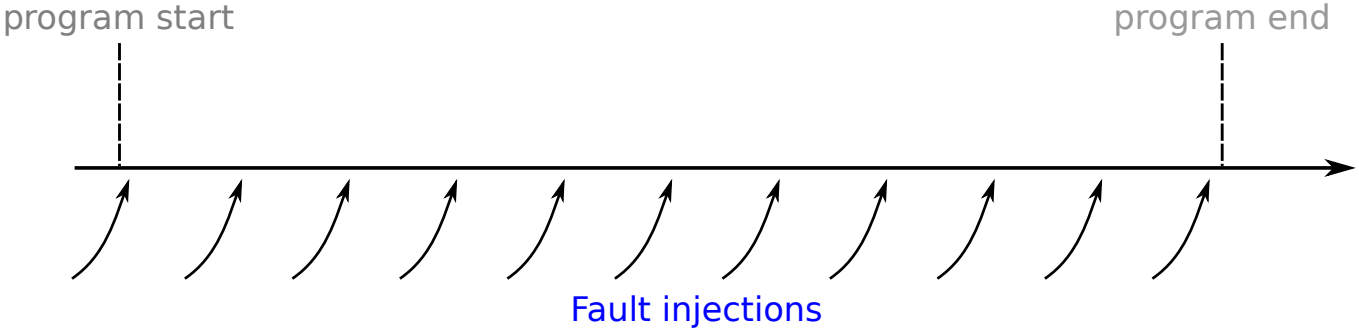
Software Plagiarism in Embedded Systems

- A product comes to the market with the same capabilities
- *Does the system contain our intellectual property?*



- Adversary takes our binary
- Effective read-out protection
- Comparison of code binaries not possible
- *Our solution*: compare fault channel leakage of the two implementations

Our Approach: Use the Fault Side Channel



1. Profile fault channel leakage
   - A fault scan of the entire implementation
   - Try inducing a fault in each clock cycle
   - Observe the output and convert into a string
     - 0: output as expected—no fault has occurred
     - 1: output wrong—fault has occurred
     - 2: program crash
   - Assumption: We should be able to distinguish faulty outputs from non-faulty outputs
2. Compare two profiles and make a decision
   - Normalized edit distance to compare two strings

-> *No need to insert a watermark—the fault channel leakage serves as the code's own watermark*

Edit Distance Between Two Strings

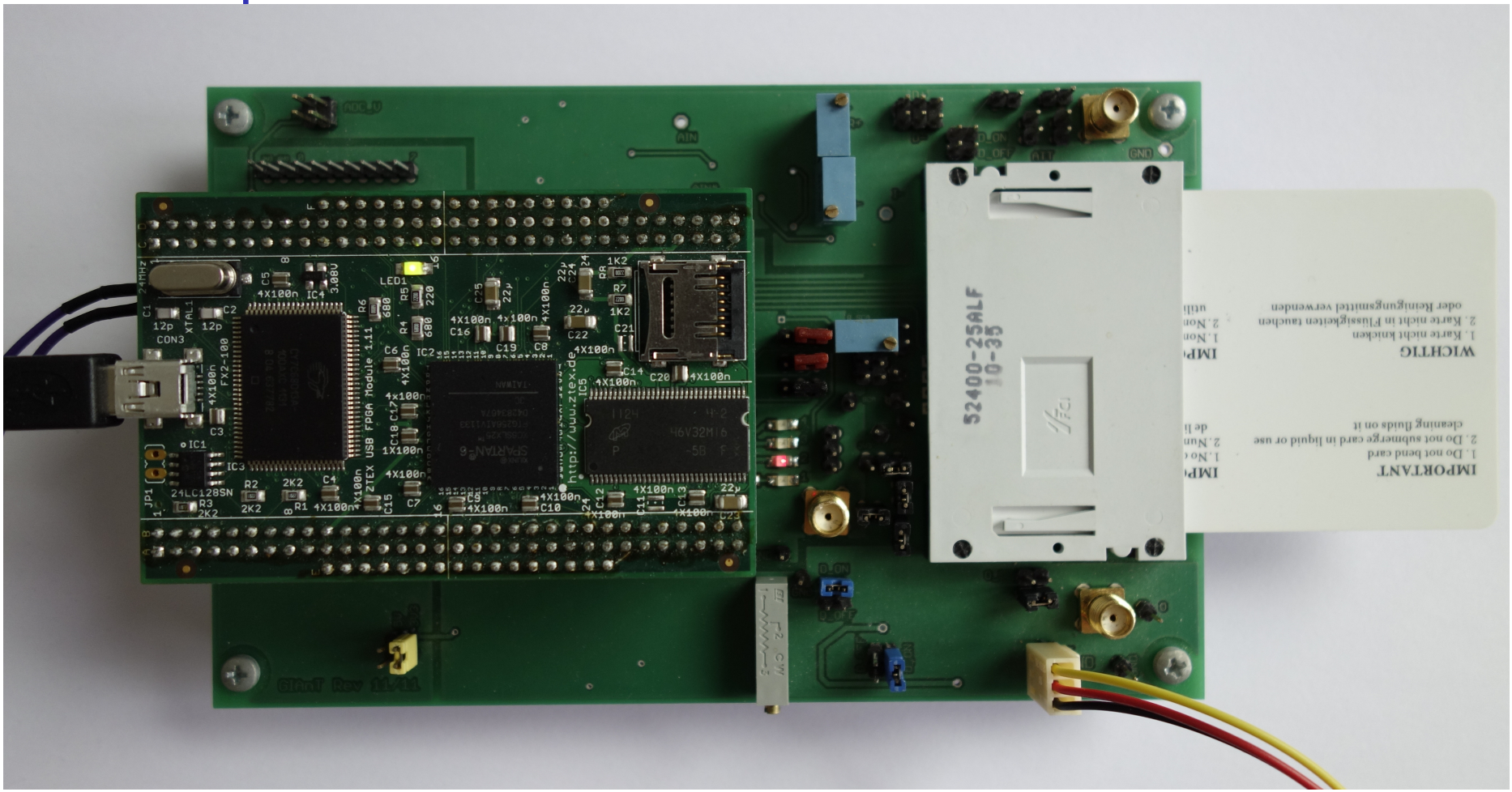
- What is the cost of transforming s_1 into s_2?
  - insert (cost 1)
  - delete (cost 1)
  - substitute (cost 1)

		A	t	e	B	s	t	C
	0	**1**	2	3	4	5	6	7
t	1	1	**1**	2	3	4	5	6
e	2	2	2	**1**	**2**	3	4	5
s	3	3	3	2	2	**2**	3	4
t	4	4	3	3	3	3	**2**	**3**

		t	e	s	t	A	B	C
	0	1	2	3	4	5	6	7
t	1	**0**	1	2	3	4	5	6
e	2	1	**0**	1	2	3	4	5
s	3	2	1	**0**	1	2	3	4
t	4	3	2	1	**0**	**1**	**2**	**3**

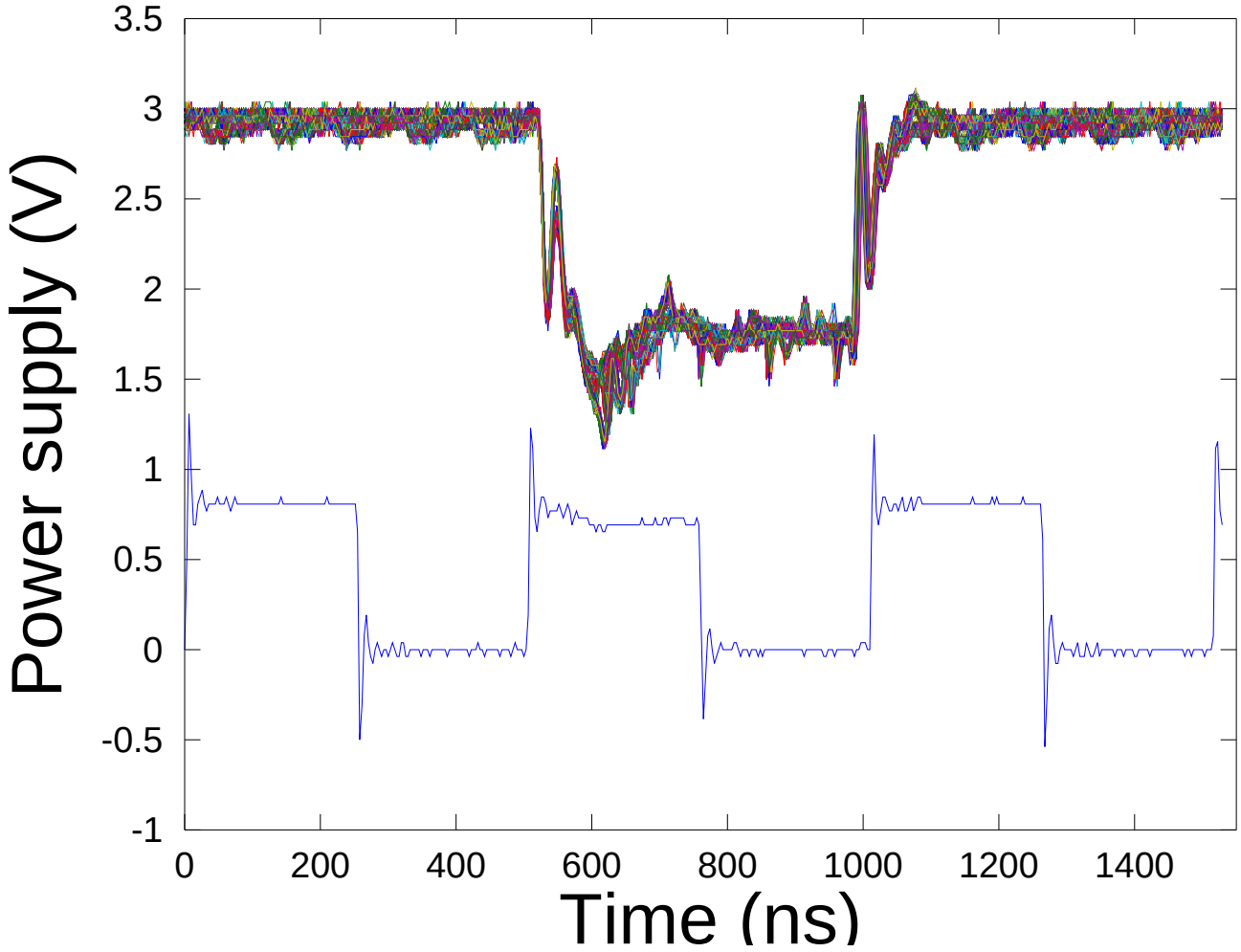
- d_e("test","AteBstC") $= 3$ (normalized 0.4286)
- d_e("test","testABC") $= 3$ (normalized 0.4286)

Our Setup



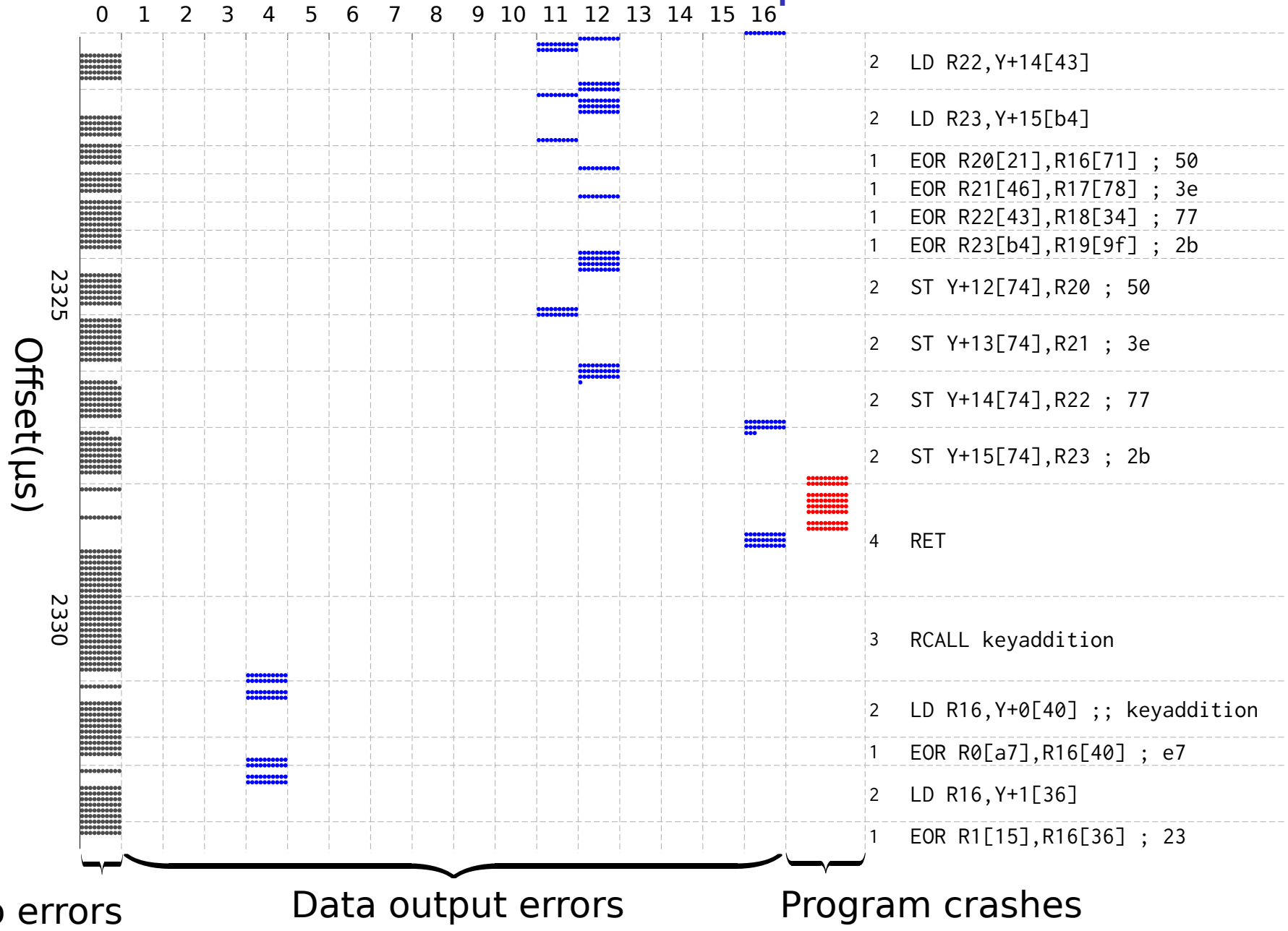
- GIAnT (Generic Implementation ANalysis Toolkit) board to induce power glitches
- Smartcard with an ATmega163 microcontroller running at 2MHz

Fault Injection with the GIAnT Board

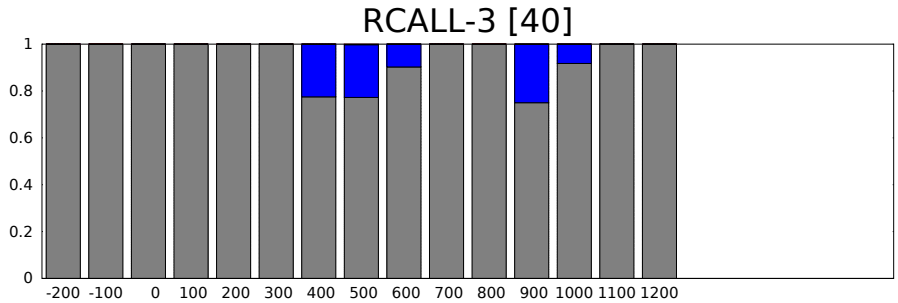


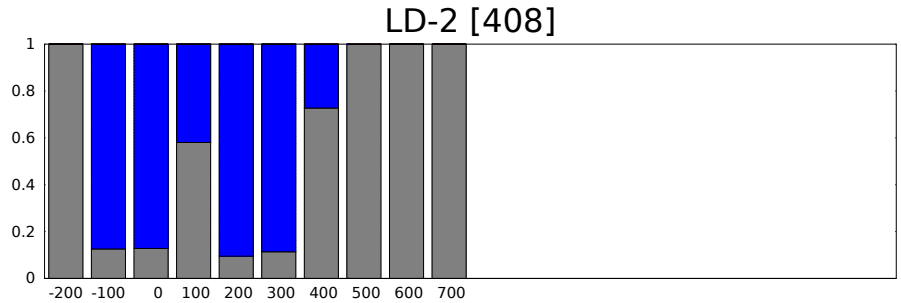
- Injection offset
- Injection pulse width

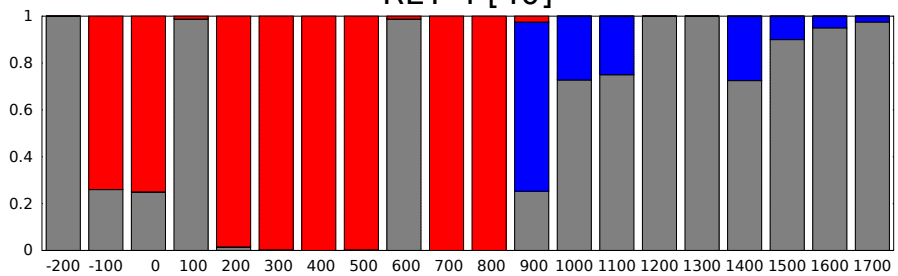
10 Fault Scans of an AES 128 Implementation

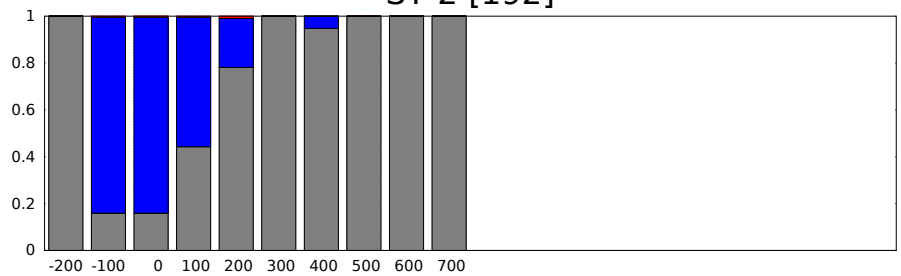


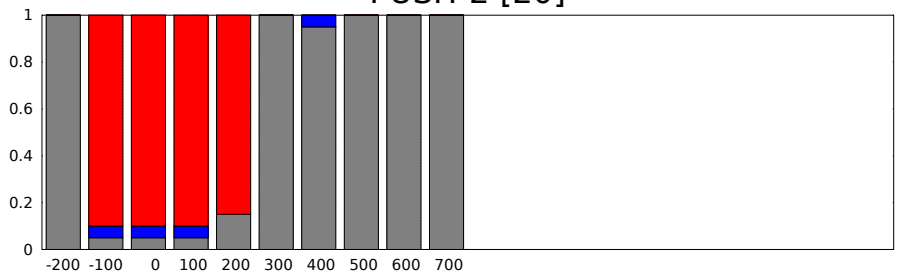
Fault Sensitivity of Instructions

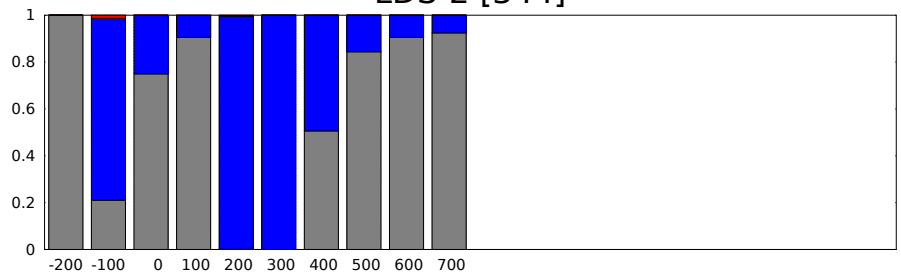


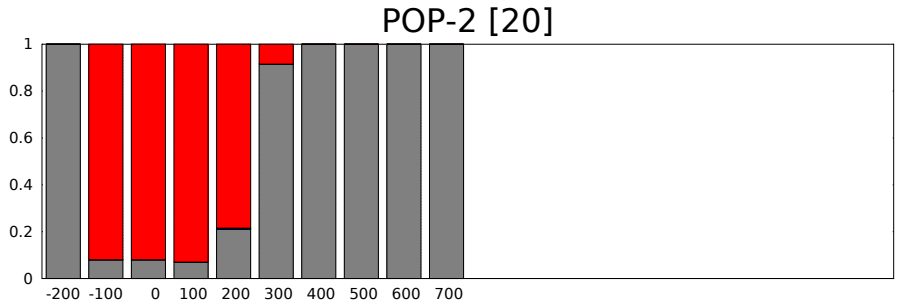


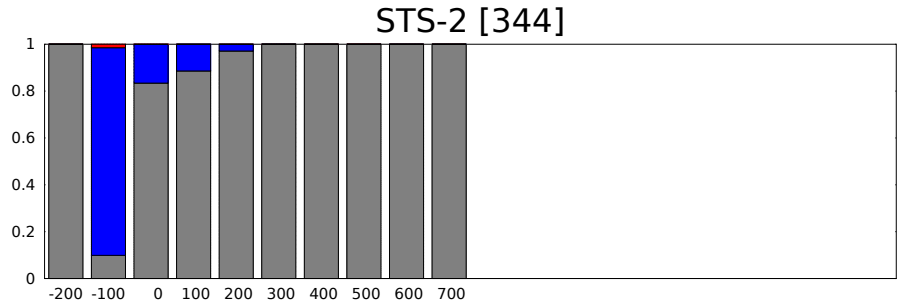












Test Applications and Experiments Overview

Implementation	AES0	AES1 v0	AES1 v1	AES1 v2	AES2 v0	AES2 v1	AES2 v2
Language	assembly	assembly	assembly	assembly	C	C	C
Optimization	-	-	-	-	-O3	-O3	-O2
Compiler version	-	-	-	-	4.8.4	4.3.3	4.3.3
N. of clock cycles	5705	4480	4480	5569	12010	12006	21980
N. of instructions	15	28	28	32	38	32	38
Inj. step size	100 ns	100 ns	500 ns	500 ns	500 ns	500 ns	500 ns
Inj. pulse width	500 ns	500 ns	500 ns	500 ns	500 ns	500 ns	500 ns
N. of scans	10	10	5	5	10	10	10
All key bytes	0x0a	0x0a	random	0x0a	0x0a	0x0a	0x0a
All plaintext bytes	0x09	0x09	random	0x09	0x09	0x09	0x09

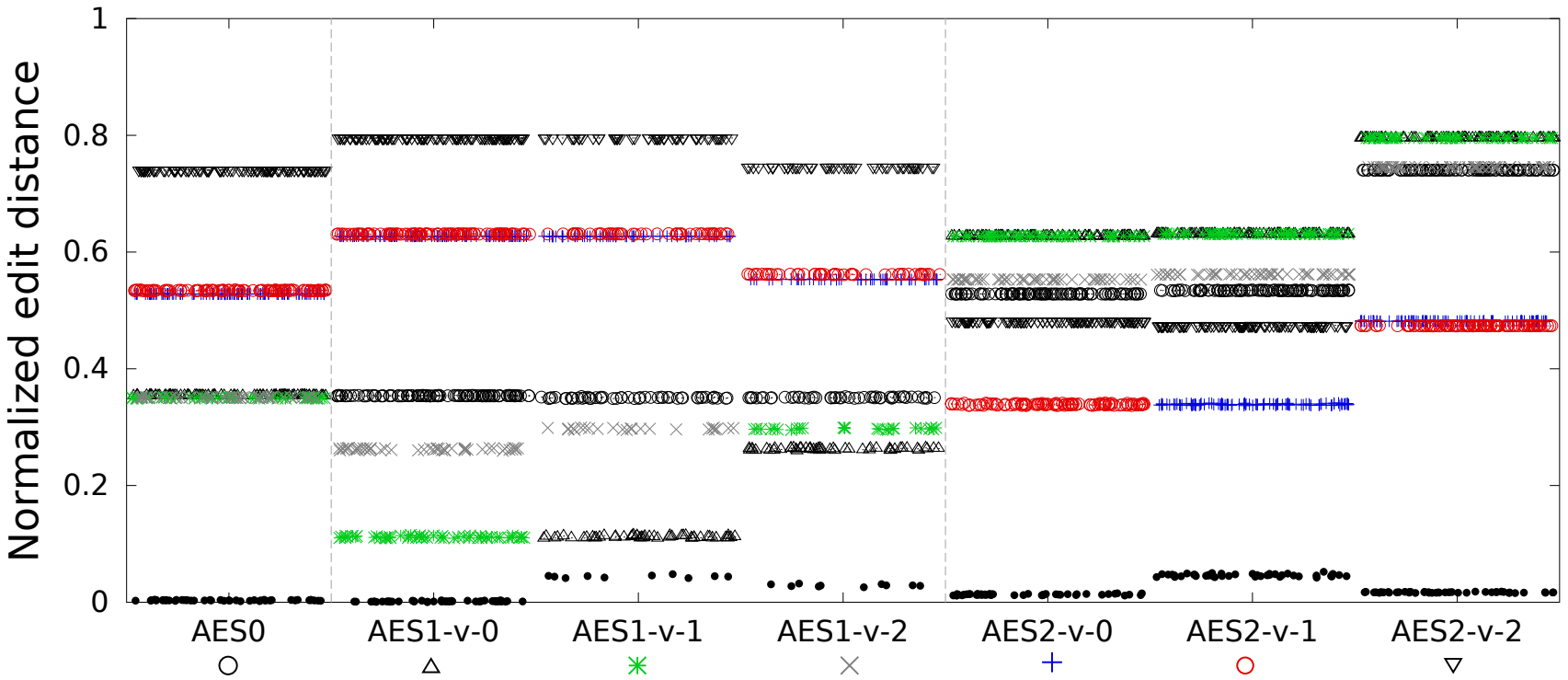
- Experiments
  - Repeatability
  - Multiple traces—using a majority string
  - Comparing the same implementations
  - Comparing different implementations
  - Comparing modified versions of the same implementation

Experiments: Repeatability and Majority String

- Repeatability
  - AES0 (28550 FIs): $\overline{d_e}(S_i, S_j) \approx 62.8 \pm 6.1$
  - AES1-v-0 (22500 FIs): $\overline{d_e}(S_i, S_j) \approx 41.6 \pm 5.3$
- Majority string

Impl.	No. fault injections	$\overline{d_e}(S_i, S_j)$	$\overline{d_e}(S_i, \overline{S})$
AES0	28550	62.8 ± 6.1	38.0 ± 6.4
AES1-v-0	22500	41.6 ± 5.3	26.7 ± 4.5

Experiments: Cross-Comparison



	AES0	AES1-v-0	AES1-v-1	AES1-v-2	AES2-v-0	AES2-v-1	AES2-v-2
AES0	**0.0032**	0.3537	0.3502	0.3506	0.5281	0.5342	0.7404
AES1-v-0	0.3537	**0.0015**	0.1116	0.2623	0.6272	0.6307	0.7954
AES1-v-1	0.3502	0.1116	**0.0441**	0.2972	0.6269	0.6309	0.7954
AES1-v-2	0.3506	0.2623	0.2972	**0.0288**	0.5529	0.5617	0.7454
AES2-v-0	0.5281	0.6272	0.6269	0.5529	**0.0131**	0.3389	0.4815
AES2-v-1	0.5342	0.6307	0.6309	0.5617	0.3389	**0.0462**	0.4738
AES2-v-2	0.7404	0.7954	0.7954	0.7454	0.4815	0.4738	**0.0169**

Related Work

- (Becker et al. 2011)
  - Embed watermarks detectable in the side channel
  - Use power consumption
  - Applicable to hardware and software
- (Durvaux et al. 2012)
  - Use power consumption as its own watermark
  - Applicable to hardware and software
- (Strobel et al. 2015)
  - Side channel disassembler
  - Use electromagnetic emanation
  - Detect individual instructions
  - Applicable to software

Summary

- Method to detect plagiarized assembly code
- Perform fault scans of the entire implementations
- Compare the fault scans using normalized edit distance

- Future Work
  - Global **and** local matching to find subparts of similar code
  - Application to hardware (FPGAs)

Thanks for listening

Any questions?