

Detecting Similar Code Segments through Side Channel Leakage in Microcontrollers

Peter Samarin^{1,2} and Kerstin Lemke-Rust¹

Bonn-Rhein-Sieg University of Applied Sciences¹
Ruhr-Universität Bochum²
Germany

November 29, 2017



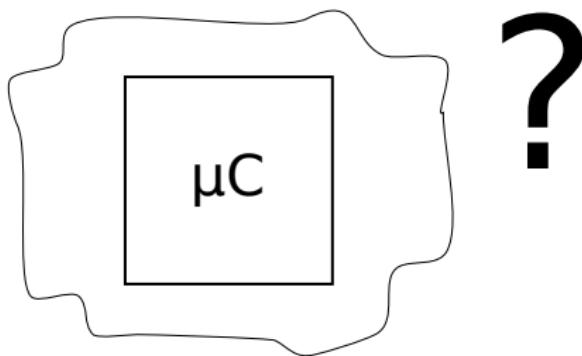
Bonn-Rhein-Sieg
University of Applied Sciences

RUHR
UNIVERSITÄT
BOCHUM

RUB

Motivation: Software Plagiarism in Microcontrollers

- ▶ 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 power side channel leakage of the two implementations

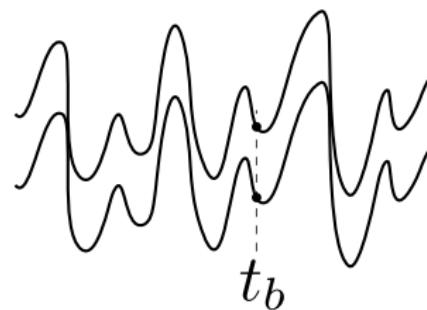
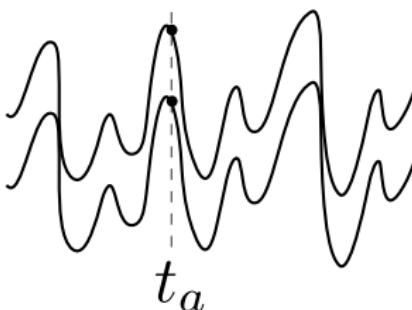
Observations about the Power Side Channel

Varying inputs

Power traces of program 1

Power traces of program 2

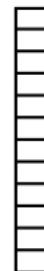
Input = x_1
⋮
Input = x_n



samples from all
traces at time t_a



samples from all
traces at time t_b



- ▶ high correlation when same data is processed
- ▶ low correlation when different data is processed

Our Approach

Varying inputs

I_1
⋮
 I_n

Power traces of
program 1



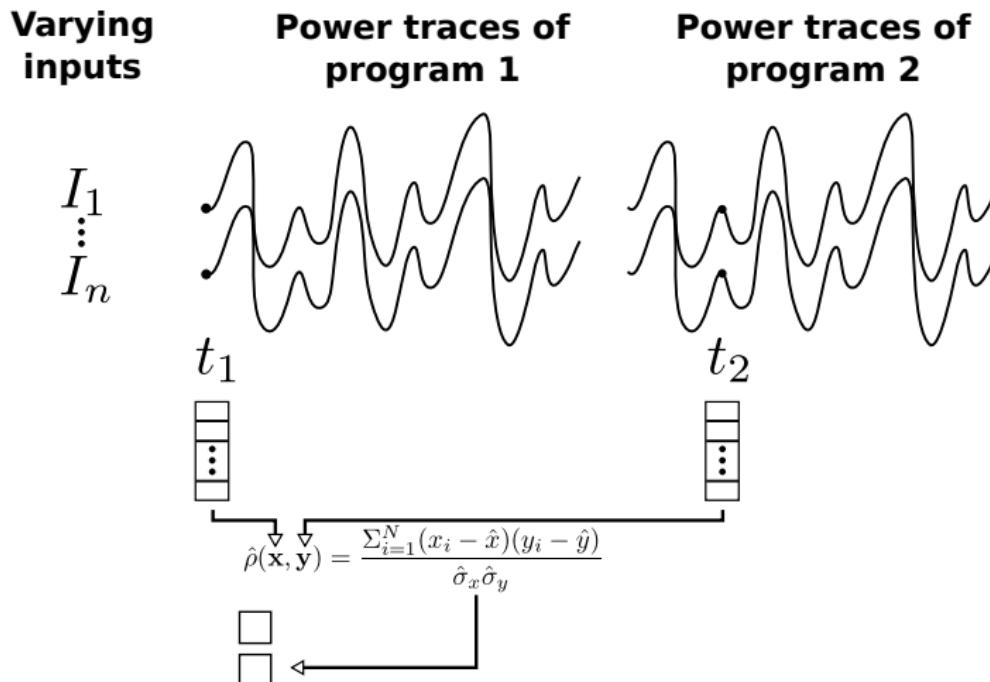
Power traces of
program 2



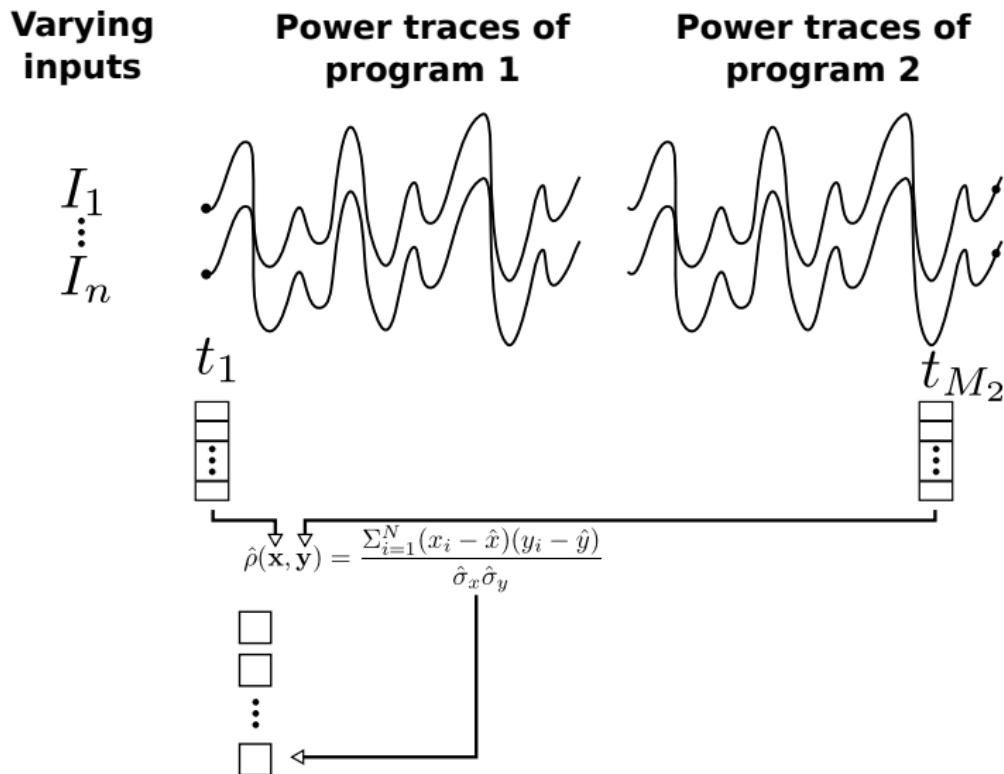
$$\hat{\rho}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^N (x_i - \hat{x})(y_i - \hat{y})}{\hat{\sigma}_x \hat{\sigma}_y}$$



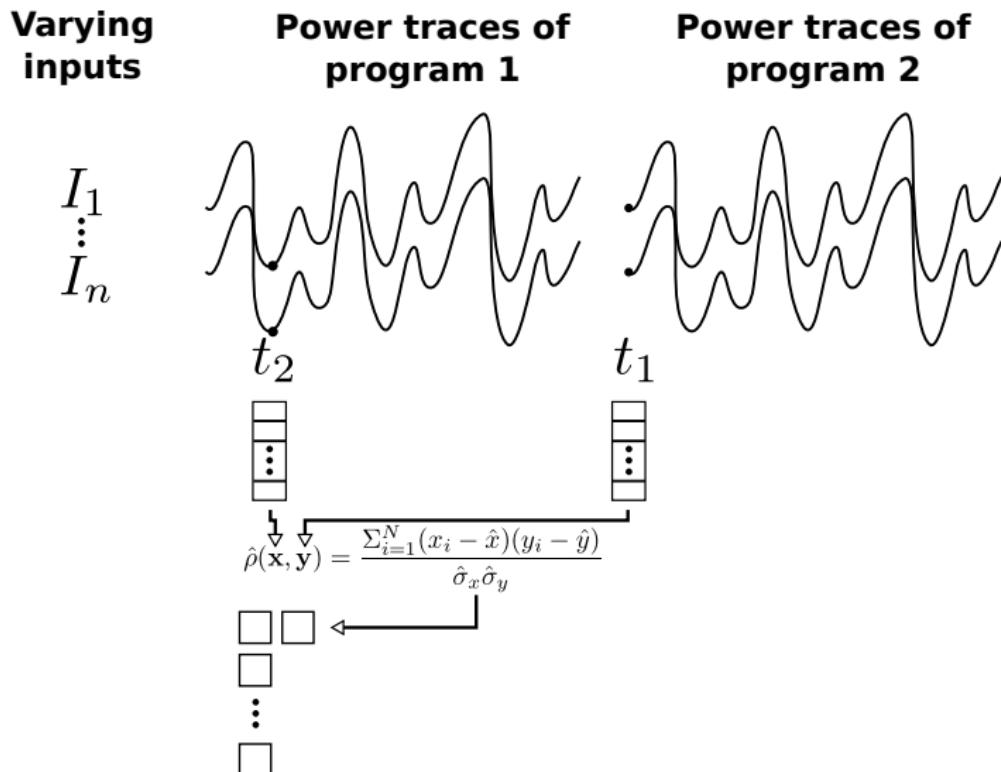
Our Approach



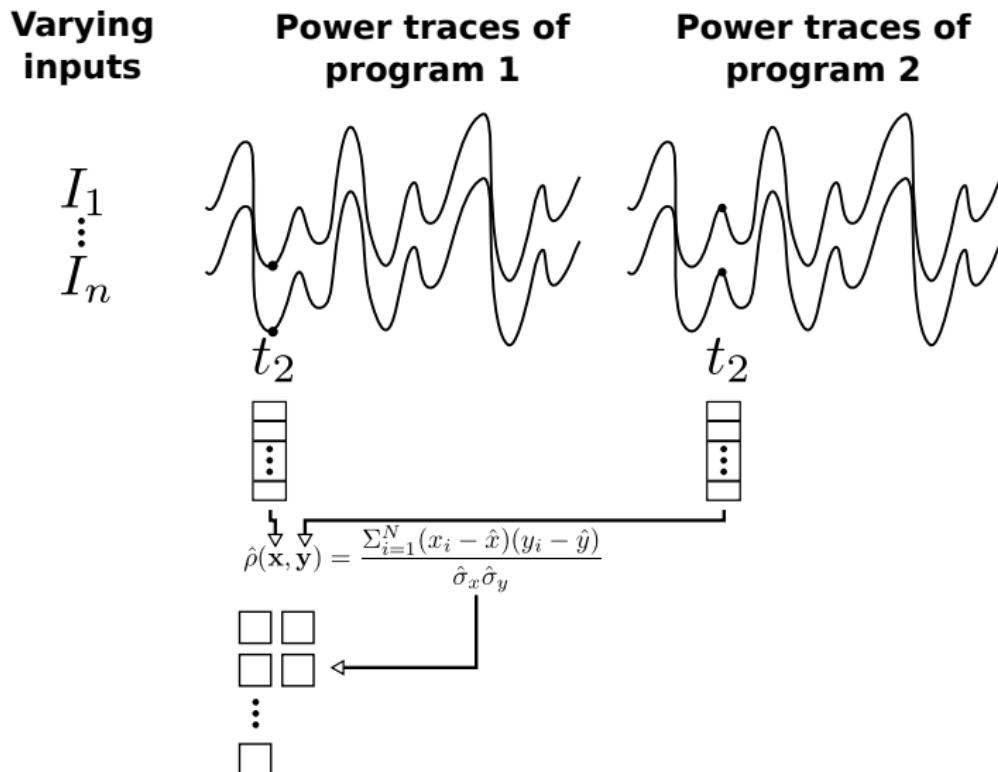
Our Approach



Our Approach



Our Approach

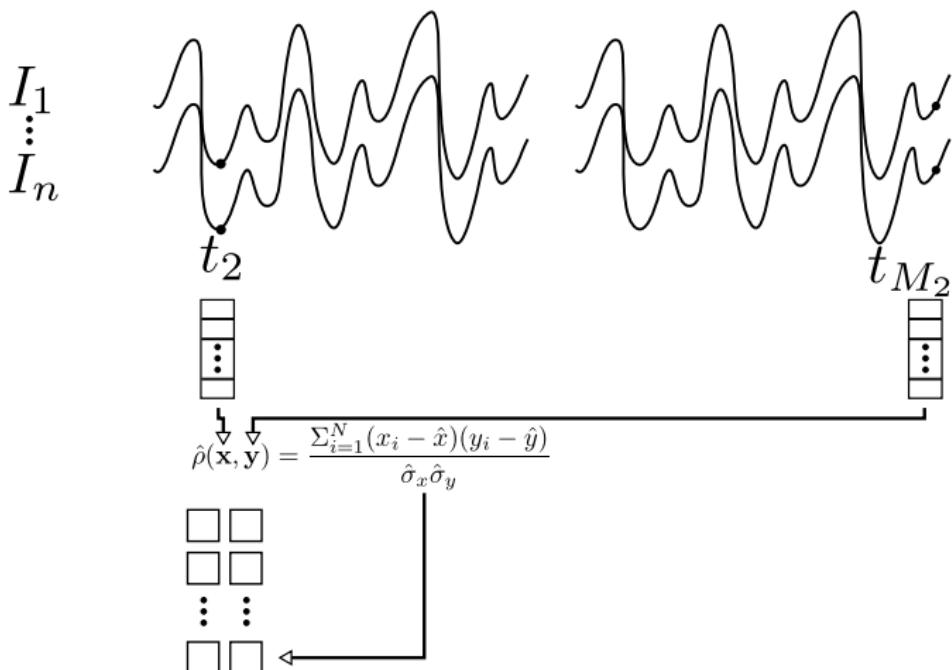


Our Approach

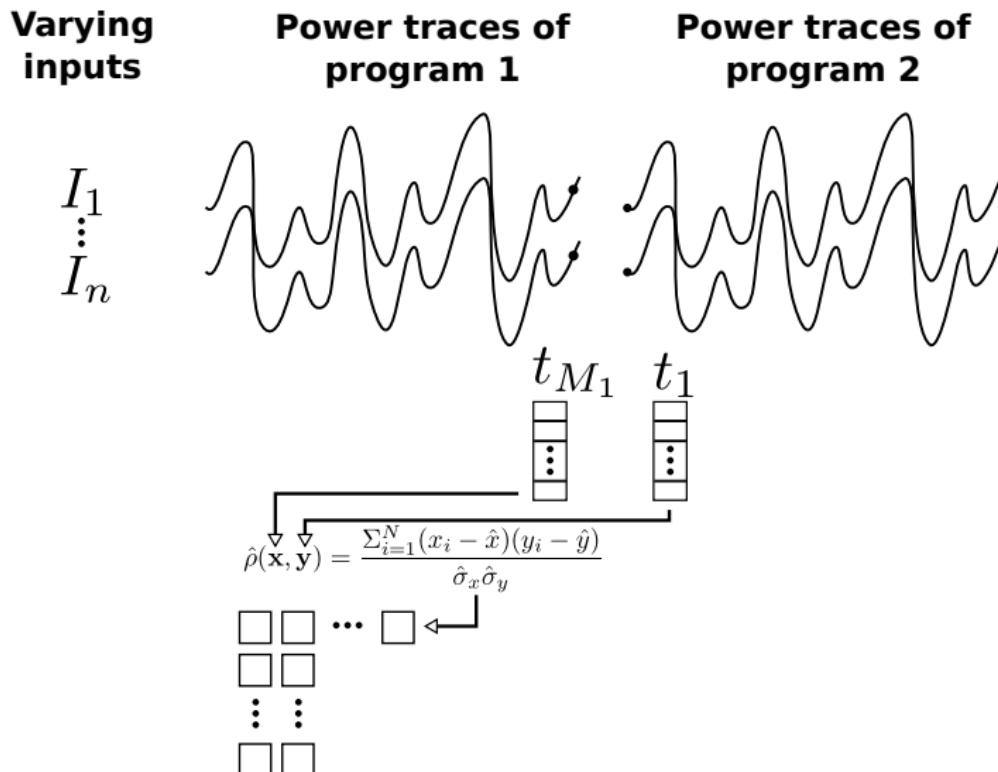
Varying inputs

Power traces of program 1

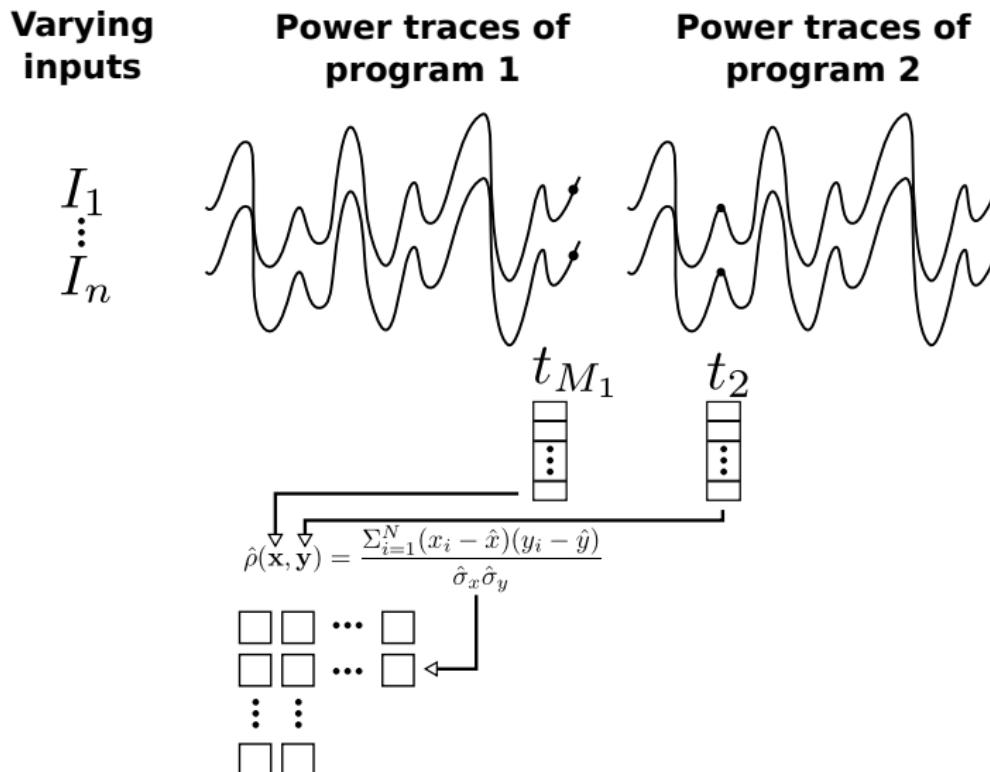
Power traces of program 2



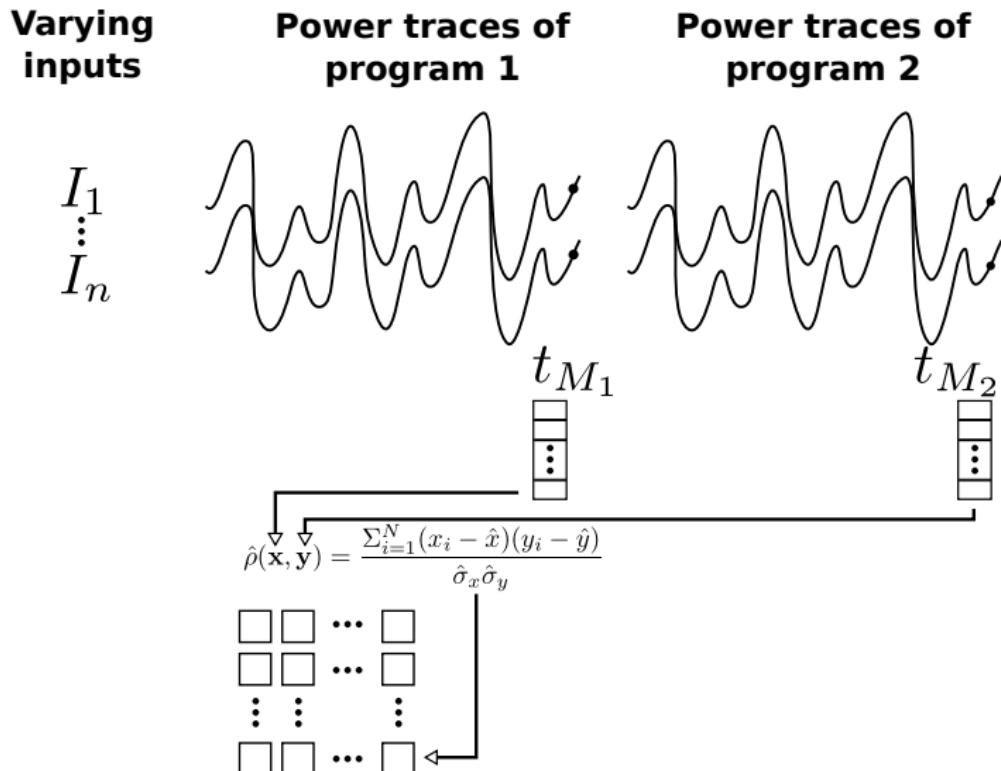
Our Approach



Our Approach



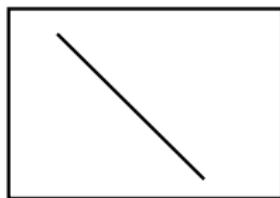
Our Approach: Correlate at all Times



Expectations about the Similarity Matrix

- ▶ The similarity matrix shows at what time similar computations happen

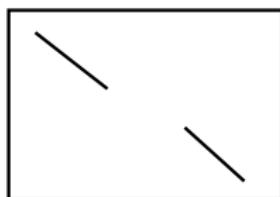
Identical program,
identical data



Similar program,
similar data



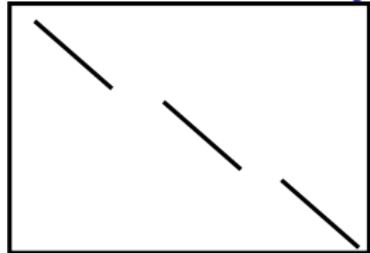
Partially identical program,
identical data



Different program
or
different data

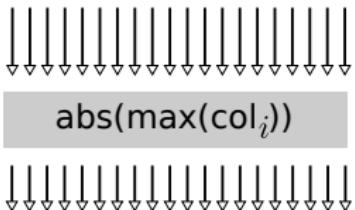


Our Approach: Similarity measure



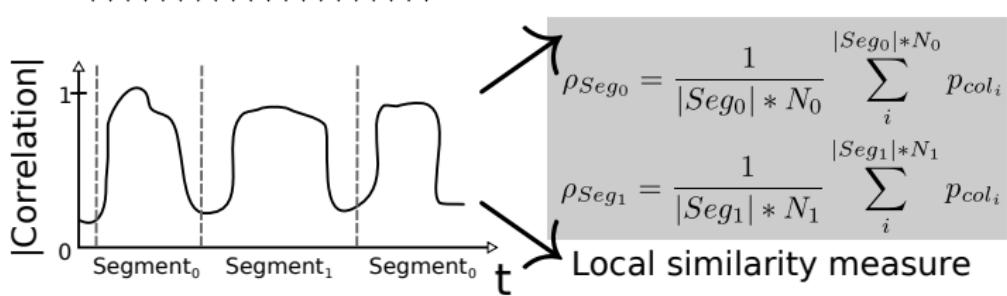
Genuine program

Suspicious program



$$\rho_{col} = \frac{1}{M_1} \sum_{i=1}^{M_1} p_{col_i}$$

Global similarity measure

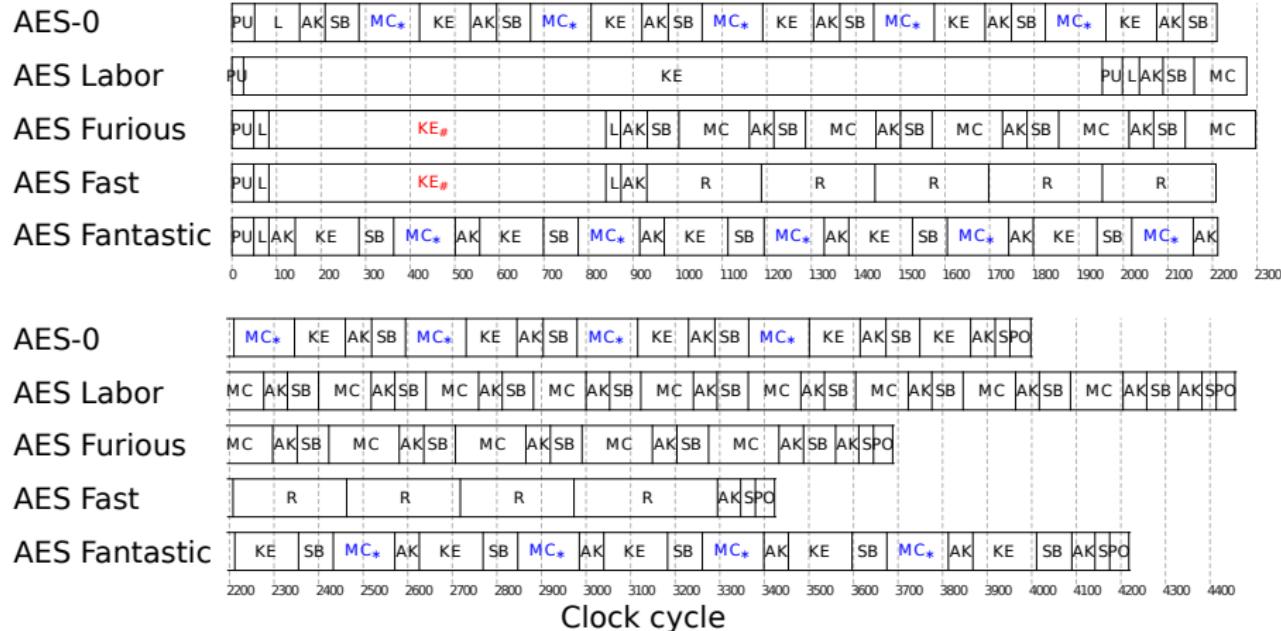


Experimental Setup

- ▶ Smartcards with ATMega163 microcontroller
 - ▶ 8-bit μC , running at 4MHz
- ▶ Measure using a digital oscilloscope (PicoScope 6402C)
 - ▶ sampling rate is 375 MHz



Test Programs: Implementations of AES in Assembly



PU - push registers

PO - pop registers

*,# - identical code

L - load key/plaintext

S - store ciphertext

KE - key expansion

AK - add round key

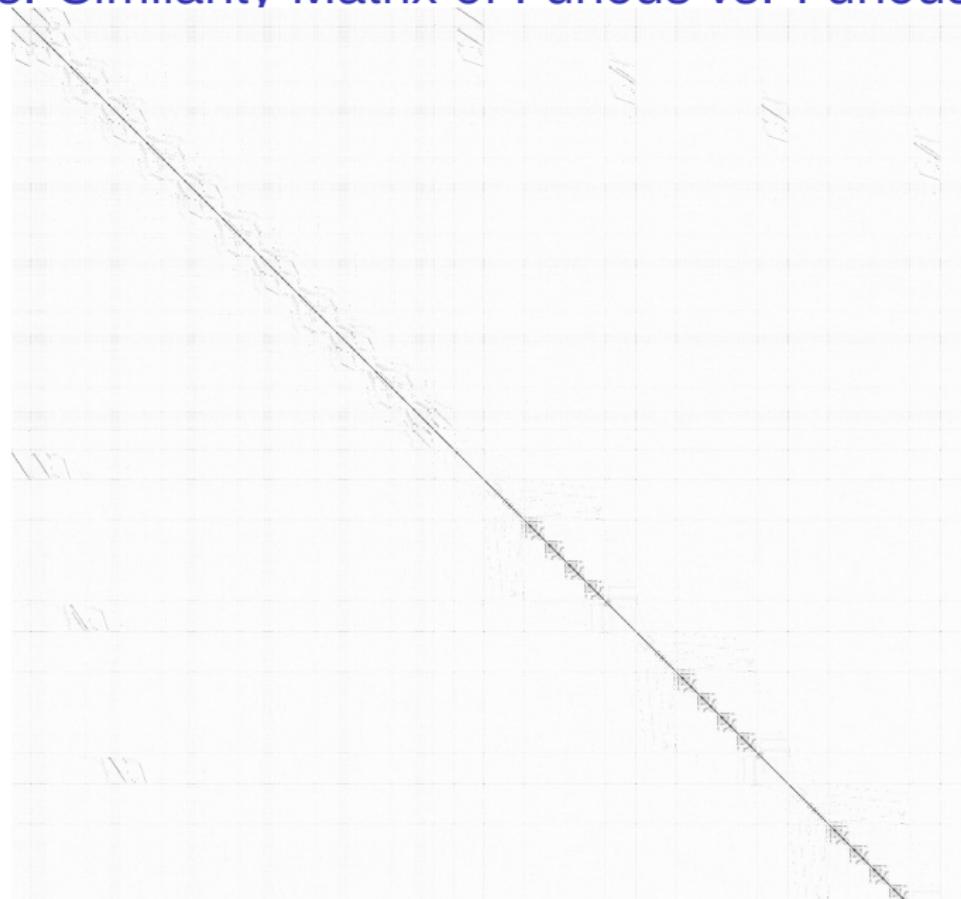
SB - shift rows and subbytes

MC - mix columns

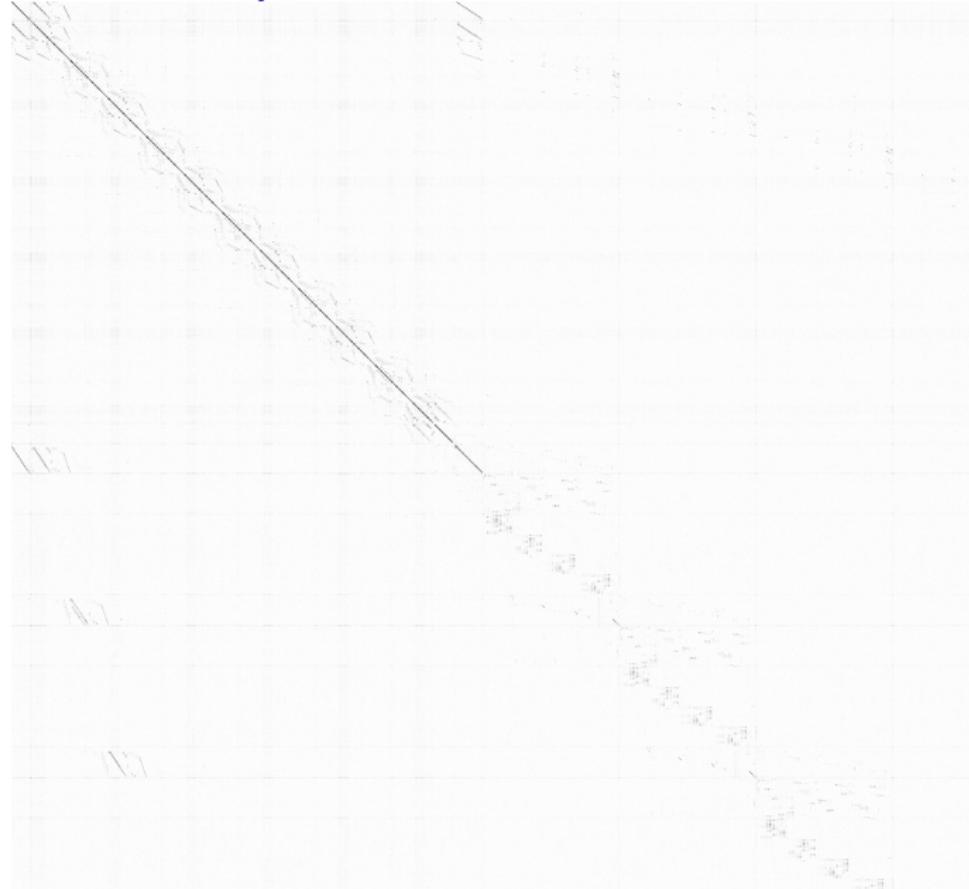
R - one AES round in *Fast*

- ▶ 10k traces were recorded for each implementation

Results: Similarity Matrix of Furious vs. Furious

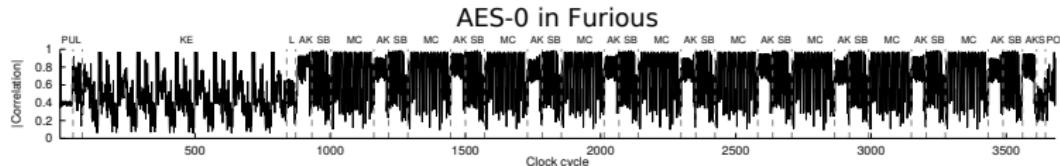
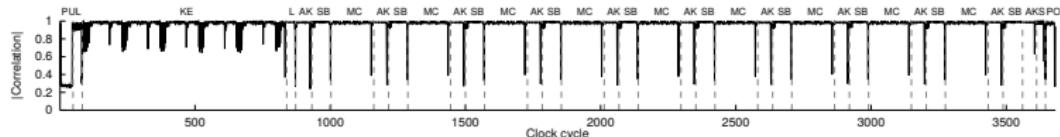


Results: Similarity Matrix of Fast vs. Furious

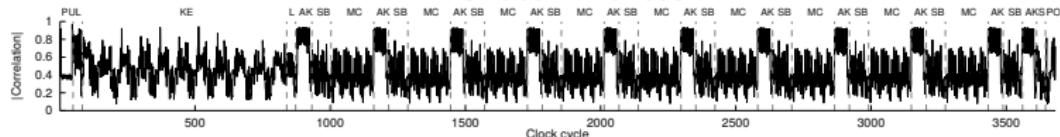


Results: Maximum Projection into Furious

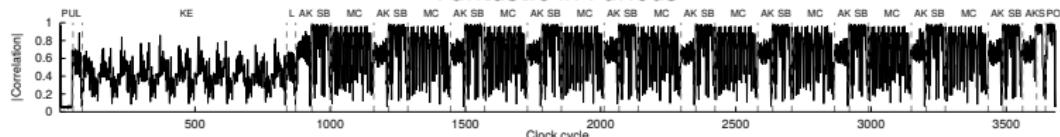
Furious in Furious



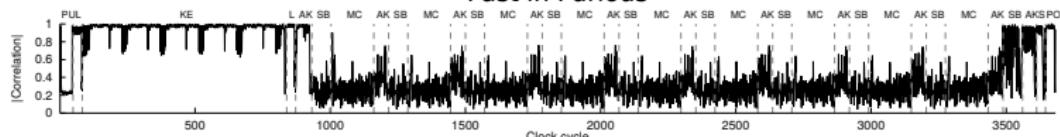
AES Labor in Furious



Fantastic in Furious



Fast in Furious



PU - push registers
PO - pop registers

L - load key/plaintext
S - store ciphertext

KE - key expansion
AK - add round key

SB - shift rows and subbytes
MC - mix columns

Results: Maximum Projection, Global Similarity

	AES-0	AES Labor	Furious	Fast	Fantastic
AES-0	0.97	0.41	0.63	0.33	0.53
AES Labor	0.42	0.91	0.46	0.29	0.39
Furious	0.61	0.44	0.96	0.45	0.54
Fast	0.35	0.32	0.46	0.96	0.29
Fantastic	0.58	0.40	0.62	0.30	0.93

Results: Maximum Projection of Code Segments

	AK	SB	MC	KE		AK	SB	MC	KE		AK	SB	MC	KE
AES-0	0.96	0.97	0.98	0.97		0.68	0.31	0.38	0.40		0.71	0.65	0.71	0.46
AES Labor	0.64	0.33	0.36	0.43		0.96	0.97	0.96	0.88		0.75	0.40	0.37	0.45
Furious	0.68	0.65	0.73	0.46		0.73	0.38	0.40	0.41		0.95	0.98	0.98	0.96
Fast	0.45	0.31	0.26	0.44		0.48	0.24	0.19	0.39		0.47	0.31	0.27	0.95
Fantastic	0.64	0.58	0.75	0.41		0.62	0.31	0.37	0.43		0.65	0.72	0.68	0.41

(a) →AES-0

(b) →AES Labor

(c) →Furious

	AK	KE	R		AK	SB	MC	KE
AES-0	0.69	0.46	0.28		0.66	0.57	0.75	0.33
AES Labor	0.73	0.45	0.23		0.62	0.32	0.35	0.40
Furious	0.85	0.95	0.27		0.62	0.71	0.70	0.32
Fast	0.97	0.95	0.98		0.43	0.27	0.25	0.31
Fantastic	0.64	0.40	0.25		0.96	0.96	0.97	0.90

(d) →Fast

(e) →Fantastic

Experiment Set #2: Furious vs. Modified Furious

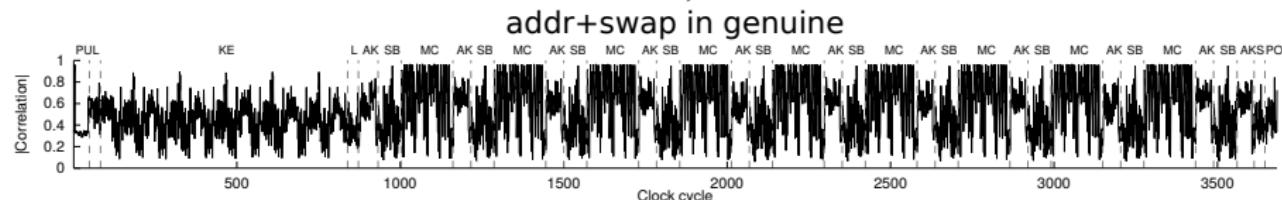
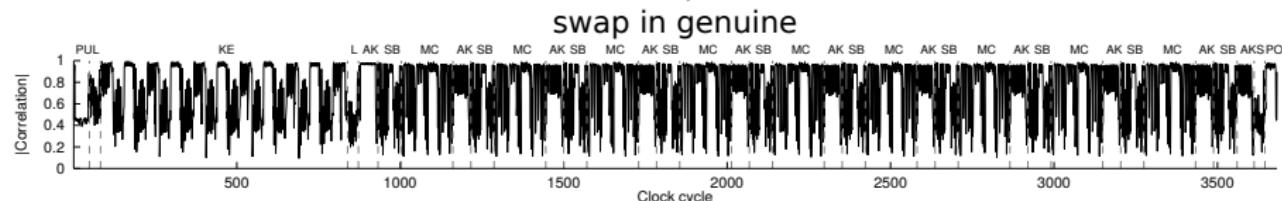
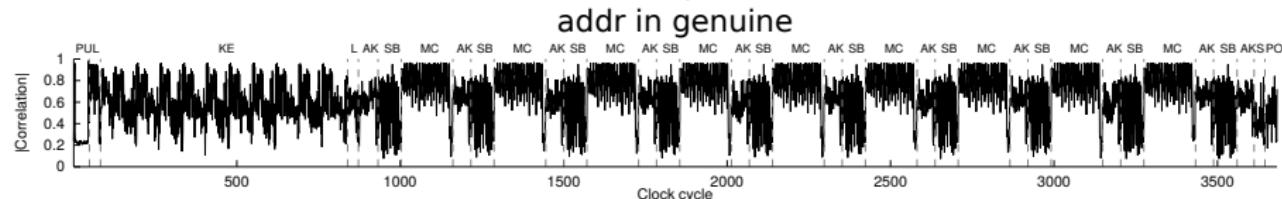
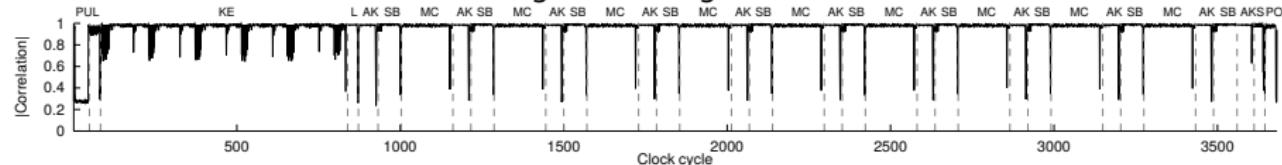
- ▶ addr: change register and data addresses
- ▶ swap: change the order of instruction execution
- ▶ addr+swap
- ▶ dummy: add 792 NOP instruction randomly
- ▶ dummy smart: add 792 leakage-generating instructions
- ▶ dummy smart+addr+swap

Dummy Smart Explanation

- ▶ Assembly language macros applied to state registers randomly throughout the code

① INC \reg DEC \reg ④ PUSH \tmp LDI \tmp, \c EOR \reg, \tmp POP \tmp	② NEG \reg NEG \reg ⑤ LDI ZL, 0x00 LPM \tmp, Z	③ ROL \reg ROR \reg ⑥ EOR \reg1, \reg2 EOR \reg2, \reg3 EOR \reg3, \reg1 EOR \tmp, \tmp	⑦ PUSH \reg1 PUSH \reg2 PUSH \reg3 POP \reg3 POP \reg2 POP \reg1	⑧ MOV \tmp, \reg ; save register LDI ZH, hi8(hd_temp) LDI ZL, lo8(hd_temp) LD \reg, z MOV \reg, \tmp ; restore register
---	---	--	--	--

Results: Maximum Projection genuine in genuine



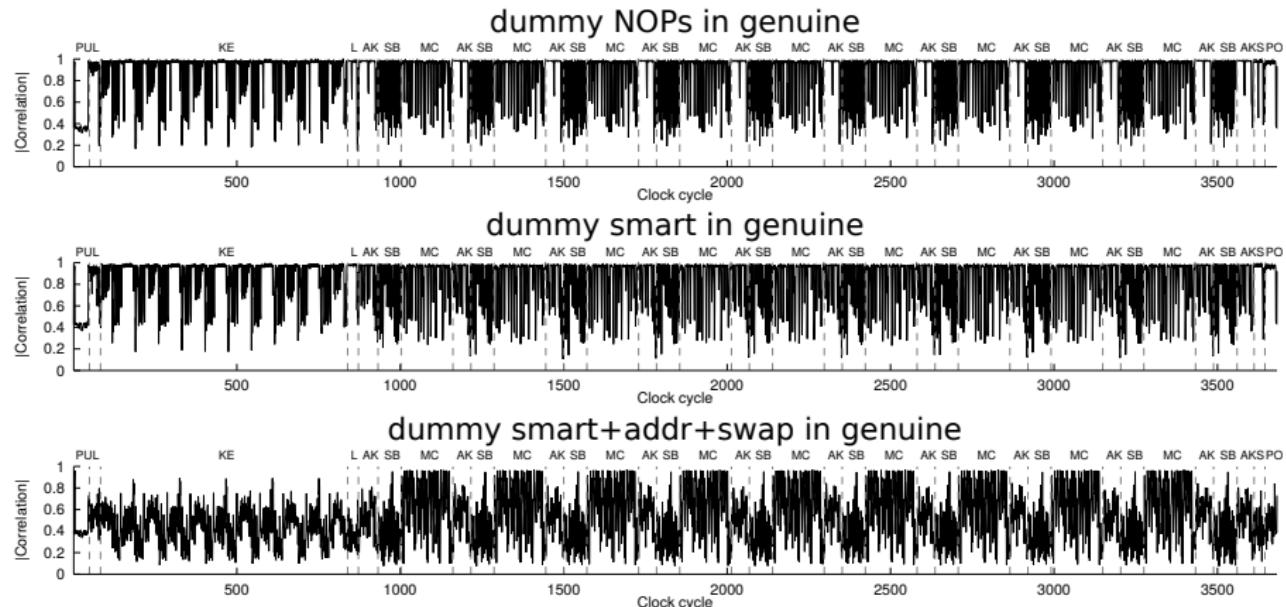
PU - push registers
PO - pop registers

L - load key/plaintext
S - store ciphertext

KE - key expansion
AK - add round key

SB - shift rows and subbytes
MC - mix columns

Results: Maximum Projection Contd.



PU - push registers
PO - pop registers

L - load key/plaintext
S - store ciphertext

KE - key expansion
AK - add round key

SB - shift rows and subbytes
MC - mix columns

Results: Detection of Similar Code Segments

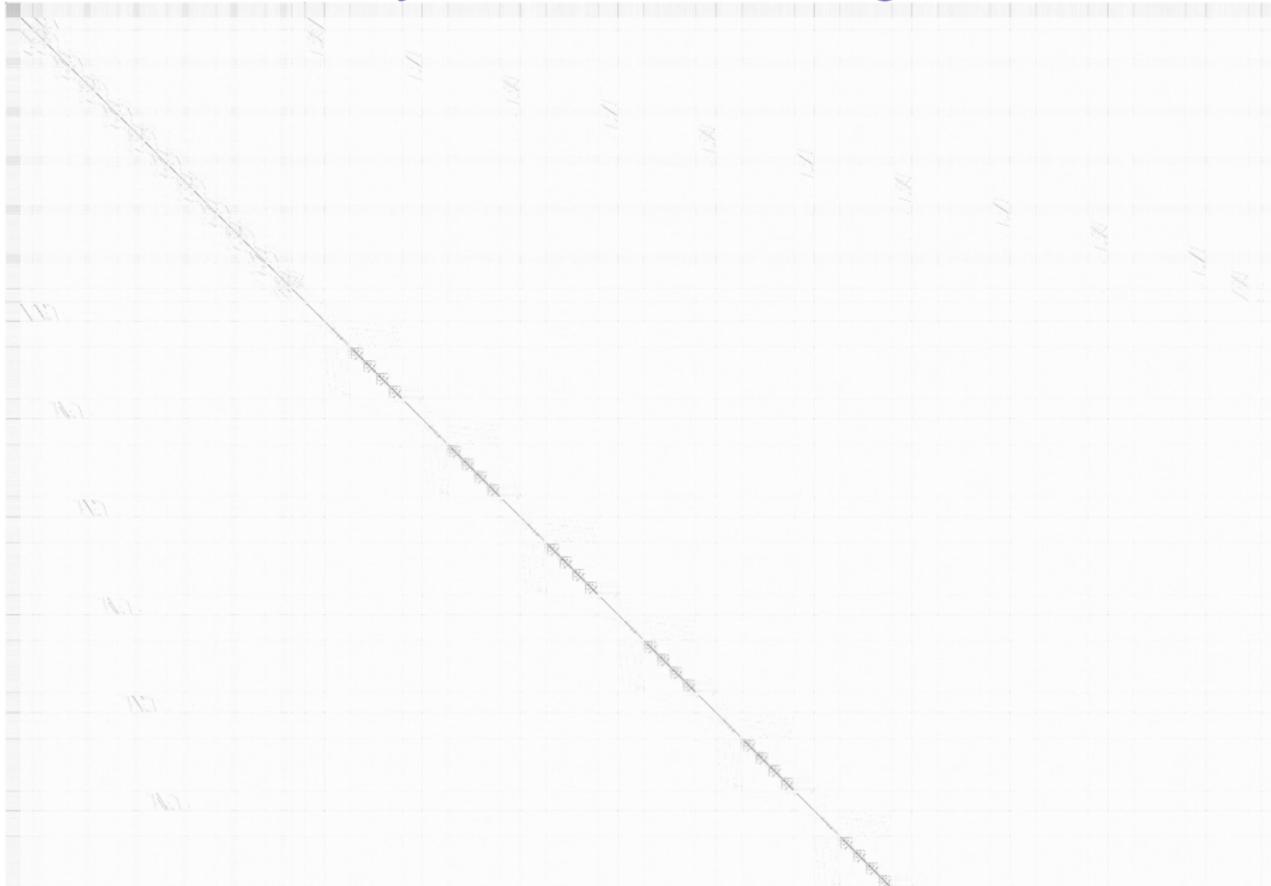
	genuine	AK	SB	MC	KE
genuine	0.96	0.95	0.98	0.98	0.96
addr	0.64	0.61	0.52	0.76	0.60
swap	0.73	0.84	0.62	0.78	0.80
addr+swap	0.52	0.59	0.37	0.64	0.45
dummy NOPs	0.84	0.92	0.72	0.87	0.86
dummy smart	0.83	0.82	0.75	0.85	0.85
dummy smart+addr+swap	0.51	0.54	0.36	0.63	0.44

(a) Global similarity

(b) Local similarity

Results: Similarity Matrix of genuine vs. genuine

Results: Similarity Matrix of addr vs. genuine



Related Work

- ▶ (Becker et al. 2011)
 - ▶ Detect Hamming weight of the instructions
 - ▶ Embed watermarks detectable in the side channel
 - ▶ Problem: not all microcontrollers leak the Hamming weight of the instruction
- ▶ (Strobel et al. 2015)
 - ▶ Side channel disassembler
 - ▶ Use electromagnetic emanation
 - ▶ Detect individual instructions
 - ▶ Problem: Only tested on one microcontroller
- ▶ (Durvaux et al. 2012)
 - ▶ Use power consumption as its own watermark
 - ▶ Horizontal correlation one two traces
 - ▶ Problem: sensitive to the dummy cycles

Conclusions and Future Work

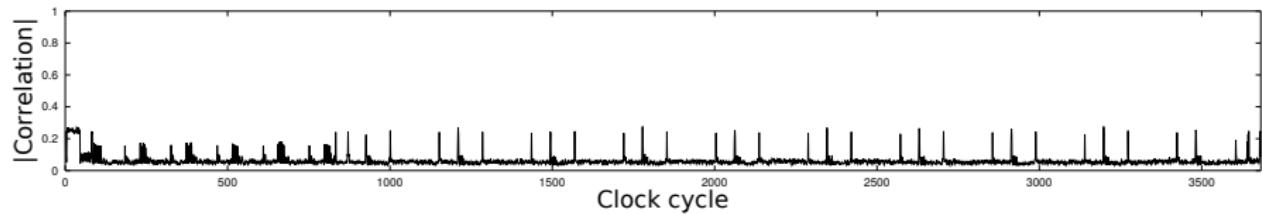
- ▶ Method for detecting similarity of programs using side channels
- ▶ We can detect identical code segments in the power consumption of a microcontroller
- ▶ Our method also works well with cases where many dummy cycles have been inserted
- ▶ Interesting application: detecting unlicensed implementations of patented technology

Future Work

- ▶ Combination of horizontal and vertical approaches
- ▶ Non-linear programs
 - ▶ dissect into data-dependent code paths
 - ▶ compute similarity for each code path
- ▶ Evaluation using different microcontrollers
- ▶ Dealing with random data

Questions?

Backup: Furious vs Furious Wrong Data



Backup: Visual Inspection

