Detecting Similar Code Segments through Side Channel Leakage in Microcontrollers

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

Bonn-Rhein-Sieg University of Applied Sciences$^1$
Ruhr-Universität Bochum$^2$
Germany

November 29, 2017
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

| \( x_1 \) | \( \vdots \) | \( x_n \) |

samples from all traces at time \( t_a \)

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

Power traces of program 2

samples from all traces at time \( t_b \)
Our Approach

Varying inputs

Power traces of program 1

Power traces of program 2

\[ I_1, \ldots, I_n \]

\[ t_1 \]

\[
\hat{\rho}(x, y) = \frac{\sum_{i=1}^{N} (x_i - \hat{x})(y_i - \hat{y})}{\hat{\sigma}_x \hat{\sigma}_y}
\]
Our Approach

Varying inputs

Power traces of program 1

Power traces of program 2

\[ I_1, I_2, \ldots, I_n \]

\[ t_1, t_2 \]

\[ \hat{\rho}(x, y) = \frac{\sum_{i=1}^{N} (x_i - \hat{x})(y_i - \hat{y})}{\hat{\sigma}_x \hat{\sigma}_y} \]
Our Approach

Varying inputs

\[ I_1, I_\ldots, I_n \]

Power traces of program 1

Power traces of program 2

\[ t_1, t_{M_2} \]

\[ \hat{\rho}(x,y) = \frac{\sum_{i=1}^{N} (x_i - \hat{x})(y_i - \hat{y})}{\hat{\sigma}_x \hat{\sigma}_y} \]
Our Approach

Varying inputs

\( I_1, \ldots, I_n \)

Power traces of program 1

Power traces of program 2

\[ \hat{\rho}(x, y) = \frac{\sum_{i=1}^{N} (x_i - \hat{x})(y_i - \hat{y})}{\hat{\sigma}_x \hat{\sigma}_y} \]
Our Approach

Varying inputs

\[ I_1, \ldots, I_n \]

Power traces of program 1

Power traces of program 2

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

Peter Samarin and Kerstin Lemke-Rust
Detecting Similar Code Segments through Side Channel Leakage
Our Approach

Varying inputs

Power traces of program 1

Power traces of program 2

\[ I_1 \]

\[ I_n \]

\[ t_2 \]

\[ t_{M_2} \]

\[ \rho(x, y) = \frac{\sum_{i=1}^{N} (x_i - \hat{x})(y_i - \hat{y})}{\hat{\sigma}_x \hat{\sigma}_y} \]
Our Approach

Varying inputs

Power traces of program 1

Power traces of program 2

\[ I_1 \]

\[ I_n \]

\[ t_{M_1} \]

\[ t_1 \]

\[ \hat{\rho}(x, y) = \frac{\sum_{i=1}^{N} (x_i - \hat{x})(y_i - \hat{y})}{\hat{\sigma}_x \hat{\sigma}_y} \]
Our Approach

Varying inputs

\[ I_1 \]
\[ I_2 \]
\[ I_n \]

Power traces of program 1

Power traces of program 2

\[ \hat{\rho}(x,y) = \frac{\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})}{\hat{\sigma}_x \hat{\sigma}_y} \]
Our Approach: Correlate at all Times

Varying inputs

Power traces of program 1

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

Power traces of program 2

$I_1$
$I_2$
$\vdots$
$I_n$
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

|Correlation|

Segment₀  Segment₁  Segment₀

Suspicious program

Genuine program

Global similarity measure

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

Local similarity measure

$$\rho_{Seg₀} = \frac{1}{|Seg₀| \times N₀} \sum_i p_{col_i}$$

$$\rho_{Seg₁} = \frac{1}{|Seg₁| \times N₁} \sum_i p_{col_i}$$
Experimental Setup

- Smartcards with ATmega163 microcontroller
  - 8-bit $\mu C$, running at 4MHz
- Measure using a digital oscilloscope (PicoScope 6402C)
  - Sampling rate is 375 MHz
Test Programs: Implementations of AES in Assembly

<table>
<thead>
<tr>
<th>Implementation</th>
<th>PU</th>
<th>L</th>
<th>AK</th>
<th>SB</th>
<th>MC*</th>
<th>KE</th>
<th>AK</th>
<th>SB</th>
<th>MC*</th>
<th>KE</th>
<th>AK</th>
<th>SB</th>
<th>MC*</th>
<th>KE</th>
<th>AK</th>
<th>SB</th>
<th>MC*</th>
<th>KE</th>
<th>AK</th>
<th>SB</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES Furious</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES Fast</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES Fantastic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Implementation | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB | MC* | KE | AK | SB |
|----------------|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|
| AES-0          |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |
| AES Labor      |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |
| AES Furious    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |
| AES Fast       |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |
| AES Fantastic  |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |     |    |    |    |

Clock cycle

- 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

Detecting Similar Code Segments through Side Channel Leakage
Results: Similarity Matrix of Fast vs. Furious
Results: Maximum Projection into Furious

Furious in Furious

 AES-0 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

<table>
<thead>
<tr>
<th></th>
<th>AES-0</th>
<th>AES Labor</th>
<th>Furious</th>
<th>Fast</th>
<th>Fantastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-0</td>
<td>0.97</td>
<td>0.41</td>
<td>0.63</td>
<td>0.33</td>
<td>0.53</td>
</tr>
<tr>
<td>AES Labor</td>
<td>0.42</td>
<td>0.91</td>
<td>0.46</td>
<td>0.29</td>
<td>0.39</td>
</tr>
<tr>
<td>Furious</td>
<td>0.61</td>
<td>0.44</td>
<td>0.96</td>
<td>0.45</td>
<td>0.54</td>
</tr>
<tr>
<td>Fast</td>
<td>0.35</td>
<td>0.32</td>
<td>0.46</td>
<td>0.96</td>
<td>0.29</td>
</tr>
<tr>
<td>Fantastic</td>
<td>0.58</td>
<td>0.40</td>
<td>0.62</td>
<td>0.30</td>
<td>0.93</td>
</tr>
</tbody>
</table>
Results: Maximum Projection of Code Segments

<table>
<thead>
<tr>
<th></th>
<th>AK</th>
<th>SB</th>
<th>MC</th>
<th>KE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-0</td>
<td>0.96</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>AES Labor</td>
<td>0.64</td>
<td>0.33</td>
<td>0.36</td>
<td>0.43</td>
</tr>
<tr>
<td>Furious</td>
<td>0.68</td>
<td>0.65</td>
<td>0.73</td>
<td>0.46</td>
</tr>
<tr>
<td>Fast</td>
<td>0.45</td>
<td>0.31</td>
<td>0.26</td>
<td>0.44</td>
</tr>
<tr>
<td>Fantastic</td>
<td>0.64</td>
<td>0.58</td>
<td>0.75</td>
<td>0.41</td>
</tr>
</tbody>
</table>

(a) → AES-0

<table>
<thead>
<tr>
<th></th>
<th>AK</th>
<th>SB</th>
<th>MC</th>
<th>KE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-0</td>
<td>0.68</td>
<td>0.31</td>
<td>0.38</td>
<td>0.40</td>
</tr>
<tr>
<td>AES Labor</td>
<td>0.96</td>
<td>0.97</td>
<td>0.96</td>
<td>0.88</td>
</tr>
<tr>
<td>Furious</td>
<td>0.73</td>
<td>0.38</td>
<td>0.40</td>
<td>0.41</td>
</tr>
<tr>
<td>Fast</td>
<td>0.48</td>
<td>0.24</td>
<td>0.19</td>
<td>0.39</td>
</tr>
<tr>
<td>Fantastic</td>
<td>0.62</td>
<td>0.31</td>
<td>0.37</td>
<td>0.43</td>
</tr>
</tbody>
</table>

(b) → AES Labor

<table>
<thead>
<tr>
<th></th>
<th>AK</th>
<th>SB</th>
<th>MC</th>
<th>KE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-0</td>
<td>0.71</td>
<td>0.65</td>
<td>0.71</td>
<td>0.46</td>
</tr>
<tr>
<td>AES Labor</td>
<td>0.75</td>
<td>0.40</td>
<td>0.37</td>
<td>0.45</td>
</tr>
<tr>
<td>Furious</td>
<td>0.95</td>
<td>0.98</td>
<td>0.98</td>
<td>0.96</td>
</tr>
<tr>
<td>Fast</td>
<td>0.47</td>
<td>0.31</td>
<td>0.27</td>
<td>0.95</td>
</tr>
<tr>
<td>Fantastic</td>
<td>0.65</td>
<td>0.72</td>
<td>0.68</td>
<td>0.41</td>
</tr>
</tbody>
</table>

(c) → Furious

<table>
<thead>
<tr>
<th></th>
<th>AK</th>
<th>KE</th>
<th>R</th>
<th></th>
<th>AK</th>
<th>SB</th>
<th>MC</th>
<th>KE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-0</td>
<td>0.69</td>
<td>0.46</td>
<td>0.28</td>
<td></td>
<td></td>
<td>0.66</td>
<td>0.57</td>
<td>0.75</td>
</tr>
<tr>
<td>AES Labor</td>
<td>0.73</td>
<td>0.45</td>
<td>0.23</td>
<td></td>
<td></td>
<td>0.62</td>
<td>0.32</td>
<td>0.35</td>
</tr>
<tr>
<td>Furious</td>
<td>0.85</td>
<td>0.95</td>
<td>0.27</td>
<td></td>
<td></td>
<td>0.62</td>
<td>0.71</td>
<td>0.70</td>
</tr>
<tr>
<td>Fast</td>
<td>0.97</td>
<td>0.95</td>
<td>0.98</td>
<td></td>
<td></td>
<td>0.43</td>
<td>0.27</td>
<td>0.25</td>
</tr>
<tr>
<td>Fantastic</td>
<td>0.64</td>
<td>0.40</td>
<td>0.25</td>
<td></td>
<td></td>
<td>0.96</td>
<td>0.96</td>
<td>0.97</td>
</tr>
</tbody>
</table>

(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**
Assembly language macros applied to state registers randomly throughout the code

1. INC \reg
2. NEG \reg
3. ROL \reg
4. DEC \reg
5. LDI ZL, 0x00
6. LPM \tmp, Z
7. EOR \reg, \tmp
8. MOV \tmp, \reg

PUSH \tmp
LDI \tmp, \c
EOR \reg, \tmp
POP \tmp

EOR \reg, \tmp

PUSH \reg1
PUSH \reg2
PUSH \reg3
EOR \reg1, \reg2
EOR \reg2, \reg3
EOR \reg3, \reg1
POP \reg3
POP \reg2
POP \reg1

MOV \tmp, \reg

LDI ZH, hi8(hd_temp)
LDI ZL, lo8(hd_temp)
LD \reg, Z
MOV \reg, \tmp
Results: Maximum Projection

**genuine in genuine**

<table>
<thead>
<tr>
<th>Correlation</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU L KE L AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB AK S PO</td>
<td></td>
</tr>
</tbody>
</table>

**addr in genuine**

<table>
<thead>
<tr>
<th>Correlation</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU L KE L AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB AK S PO</td>
<td></td>
</tr>
</tbody>
</table>

**swap in genuine**

<table>
<thead>
<tr>
<th>Correlation</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU L KE L AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB AK S PO</td>
<td></td>
</tr>
</tbody>
</table>

**addr+swap in genuine**

<table>
<thead>
<tr>
<th>Correlation</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU L KE L AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB MC AK SB AK S PO</td>
<td></td>
</tr>
</tbody>
</table>

- 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.

dummy NOPs in genuine

dummy smart in genuine

dummy smart+addr+swap in genuine

PU - push registers
PO - pop registers
KE - key expansion
L - load key/plaintext
AK - add round key
S - store ciphertext
SB - shift rows and subbytes
MC - mix columns
## Results: Detection of Similar Code Segments

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

(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?