Data collection and pre-processing for white-box side-channel analysis
Binary tracing

- Equivalent of an oscilloscope
  - Information collection

- White-box context
  - Perfect capture (no noise, leakage model = value)
  - Reproducible

- Debugger, emulator, instrumentation (DBIs), hardware (Intel PT)
An example: TracerGrind

- Publicly available, designed for WB analysis
  - Based on well-known Valgrind DBI
  - Focus on memory accesses
  - https://github.com/SideChannelMarvels/Tracer

- Example binary: CHES 2017 – Challenge 84

- Trace size: 1 AES ~27GB
TracerGrind limitation

❖ As said, *focus on memory accesses*

❖ What about this situation?

<table>
<thead>
<tr>
<th>Pseudo code</th>
<th>Memory</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmp = mask1_xor_val</td>
<td>⊕ mask1_xor_val</td>
<td>mask1_xor_val</td>
</tr>
<tr>
<td>tmp ^= mask1</td>
<td>⊕ mask1</td>
<td>val, mask1</td>
</tr>
<tr>
<td>tmp ^= mask2</td>
<td>⊕ mask2, ⊕ mask2_xor_val</td>
<td>mask2_xor_val, mask2</td>
</tr>
</tbody>
</table>
Working with registers

- Switch to `qemu`-based tracing to collect registers
  - `-seed 0`: stick PRNG sequence (ASLR ...)
  - `-singlestep`: dump state at each instruction rather than basic block

- Many registers, many instructions $\Rightarrow$ huge amount of data
  - Eg AES #84 generates > 120GB data 😞
  - And this is without obfuscation...

- Redundancy? Evaluate with compression
  - 120GB $\Rightarrow$ 5GB, divided by 24! (27GB $\Rightarrow$ 1.5GB for TracerGrind)
Reducing trace size: some tricks

- Deduplication
  - Noiseless data collection \(\Rightarrow\) remove redundant information
  - At the byte and bit levels

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
</tr>
</thead>
<tbody>
<tr>
<td>d0</td>
<td>57</td>
<td>d2</td>
<td>51</td>
<td>d4</td>
</tr>
<tr>
<td>1b</td>
<td>96</td>
<td>7b</td>
<td>ba</td>
<td>3d</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
</tr>
<tr>
<td>9e</td>
<td>13</td>
<td>fe</td>
<td>3f</td>
<td>b8</td>
</tr>
</tbody>
</table>

0001020304: [0, 3]
d057d251d4: [1]
1b967bba3d: [2]
9e13fe3fb8: [4]
Reducing traces size: some more tricks

- Selective dump
  - At each instruction, only collect potential changes
  - Desynchronization is an issue

- "Toward Fully Automated Analysis of Whiteboxes – Perfect Dimensionality Reduction for Perfect Leakage", Breunesse et al.
  
Deduplicating efficiently

Objectives
- fast with lowest programming effort possible
- Take advantage of HULK*

C++11 with useful features:
- Use `std::for_each`, `std::sort`
- Lambda functions

Thrust
- Replace `std::vector` by `thrust::device_vector` ⇒ automatically run on GPUs!
- Replace `std::for_each` by `thrust::for_each` ⇒ automatically parallelized!
- Compile using Threading Building Blocks backend (Intel)
- Combine with `thrust::device_vector` to run on GPUs

*HULK: 48 cores, 192GB RAM, 3TB SSD RAID, 2 GTX 1080Ti
Fun fact

- Function inlining + obfuscation: better perf than not inlined!

- Unexpected. Why?
  - Guess: obf impact smoothed when inlined, « hit-or-miss » when not
  - Security impact?
Thank you!