The Need for Speed: Applications of HPC in Side Channel Research

Dr. M. Elisabeth Oswald
Reader, EPSRC Leadership Fellow
University of Bristol
Roadmap

- Background: side channels, practical angles for research
- The BIG question: how much does my device leak?
- Summary
In case you haven’t heard of side channels ....

- Known side channels:
  - timing, power, EM
  - acoustics, de-duplication, TCP-IP traffic features, error messages, cache behaviour, ...

- Used for
  - Key recovery
  - Plaintext recovery
  - Device fingerprinting
E.g. Web traffic analysis

Profiling of web traffic allows to recover user choices even through encrypted traffic.

(Chen et al., IEEE S&P, 2010)
E.g. Web traffic analysis: features which leak

Features that leak are:
• Packet size
• Direction
• Arrival time
• TLS record lengths
• TCP acknowledg. flag
• TCP handshaking flags

Details: Mather & O., JCE 2012 (2)
Side channel research questions ...

- Are there leaks? If so what leaks? If not how can we be sure?
- How many side channel observations are needed to exploit the leaks ...?
  - One?
  - Many? (What is many?)
  - What does exploit mean? (Key recovery, partial key recovery, lambda leakage?)
- (New attacks, new countermeasures, leakage resilient crypto)
Different practical ‘angles’ for (SC) research

Distinguished by:

Degree/extent of knowledge:
- Leakage points (within a trace)
- Leakage model

Computational capabilities:
- How many leakage traces
- How much computation
**Different practical ‘angles’ for (SC) research**

Evaluator should be at least as good as best ‘practical’ attacker ...

But computational capabilities are increasing fast:

- Attack using a 32-bit key guess took just over 8 minutes in 2012 using 4 state of the art GPUs
- Same attack now takes 15 sec!
Roadmap

• Background: side channels, practical angles for research
• The BIG question: how much does my device leak?
• Summary
How to determine $\lambda$

1. Measure side channels for $N$ encryptions
2. Extract relevant data: leakage detection
3. Analyse relevant data to extract probabilities for chunks of key: leakage exploitation
4. Sift through key space using probabilities: key enumeration/rank estimation

Research question:
Given $N$ observations, how much effort is required (in 4.) to find the secret key.

Leakage bound $\lambda$
Leakage detection

Given a vector of side channel points (aka a trace, see below), determine which of the points contain leakage about a (specific) secret.

- What statistical test to use? (t-test, continuous MI, or discrete MI):
  - Genericity (i.e. it captures all sorts of leaks)
  - Computational requirements; time
  - Number of leakage traces (aka sample size)

(Power traces of AES encryption)
Leakage detection, cont.

The **better test** can spot information leakage **faster and more reliable**—it requires less data; whilst maintaining a high statistical power (i.e. probability a test correctly rejects a null hypothesis).

Can we estimate the minimum sample sizes required to achieve sufficient statistical power?

- Need to vary leakage models, noise levels, and sample sizes!!
- This is research is computationally very expensive.
Leakage detection, cont.

Heavily lifting required to evaluate effectiveness of e.g. CMI:

• Estimate MI(K;T)
• Estimate ‘zero MI’, by randomly permuting traces T (need at least around 100 permutations)
• Repeatedly ....

$$\hat{I}(K; T) = \sum_{k \in K} \int_{T} \hat{p}(k, t) \log_2 \left( \frac{\hat{p}(k, t)}{p(k)\hat{p}(t)} \right) dt.$$  

Sample size required to achieve 80% power (Toggle count leakage)

Even heavy for a single application: CMI applied to our real world AES traces demanded $2^{51}$ calls to the kernel function!
Leakage detection, cont.

Continuous MI test, high-end specification

<table>
<thead>
<tr>
<th>Option</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>550 1c x 3.3 GHz CPU</td>
<td>2021 days</td>
</tr>
<tr>
<td>550 4c x 3.3 GHz CPU</td>
<td>188.5 days</td>
</tr>
<tr>
<td>Radeon HD 6450 GPU</td>
<td>39 days</td>
</tr>
<tr>
<td>Radeon HD 7970 GPU</td>
<td>3.7 days</td>
</tr>
</tbody>
</table>

Switching to a GPU based implementation on our HPC cluster was the only way to conduct this research.
Leakage detection summary

• T-test is a good baseline test, but obviously cannot capture higher-order leaks
• CMI can be used in practice if implemented appropriately

• Bottom line: we can now assess general information leaks with some rigour!
  • See Mather & O. (et al.) Asiacrypt 2013
How to determine $\lambda$

1. Measure side channels for $N$ encryptions
2. Extract relevant data: leakage detection
3. Analyse relevant data to extract probabilities for chunks of key: leakage exploitation
4. Sift through key space using probabilities: key enumeration/rank estimation

Research question:
Given $N$ observations, how much effort is required (in 4.) to find the secret key.
Leakage exploitation

- Given a set of known leakage points what is the best strategy to exploit the leakage?
  - (How to select among the known leakage points)
  - How to combine the selected leakage points

(AES power trace)
Leakage exploitation: combining attack outcomes (AES)

**Single point attack**
- AES has 16 state bytes, assume you attack them individually:

**Combining outcomes**
- But you can attack different intermediate values, so these should be combined
Leakage exploitation, cont.

- It turned out that amalgamating distinguishing scores by `directly` using them as probabilities is a very efficient strategy.
- But working with MixColumns means we need to work with 32 bits of the key at a time. We used again a GPU based implementation, and switched to an HPC platform to do repeat experiments.

Keys attacked per second, OpenCL kernel for attacking 32 bits of key using the MixColumns operation:

<table>
<thead>
<tr>
<th>2,000 traces</th>
<th>5,000 traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i5 3550 @ 3.3 GHz, single core</td>
<td>Intel i5 3550 @ 3.3 GHz, single core</td>
</tr>
<tr>
<td>739</td>
<td>295</td>
</tr>
<tr>
<td>2x Intel Xeon E5 2670 @ 2.6 GHz, 16 cores</td>
<td>2x Intel Xeon E5 2670 @ 2.6 GHz, 16 cores</td>
</tr>
<tr>
<td>1,257,608</td>
<td>498,627</td>
</tr>
<tr>
<td>2x AMD Radeon R9 290X</td>
<td>2x AMD Radeon R9 290X</td>
</tr>
<tr>
<td>2x AMD Radeon HD 7970 GHz Edition</td>
<td>2x AMD Radeon HD 7970 GHz Edition</td>
</tr>
<tr>
<td>47,034,048</td>
<td>19,219,889</td>
</tr>
<tr>
<td>39,727,037</td>
<td>15,549,116</td>
</tr>
</tbody>
</table>
Leakage exploitation: AES column

16-bit targets, SNR = 1.0

16-bit targets, SNR = 0.0625

24-bit targets, SNR = 0.0625

32-bit targets, SNR = 0.0625
Leakage exploitation: real device
Leakage exploitation: experimental setup

- Used up to 6 workstations with 2 high end GPUs each (cost per machine around 2k GBP)
  - Both Nvidia cards and AMD
- Developed Baikal which efficiently distributes attacks across workstations and within nodes (hand threaded) utilising OpenCL
- Completed just over $2^{50}$ operations on combined distinguishing vectors in about 2 weeks
  - Details in Mather & O. (et al.) Asiacrypt 2014
Leakage exploitation summary

- Multi target attacks effectively amalgamate distinguisher outcomes of different (independently) computed attacks.
  - They can exploit multiple leakage points effectively
  - (Template attacks do not scale and so cannot be applied across large portions of leakage traces)
- Implementation is practical when appropriate hardware is used (GPUs)
**Conclusion**

HPC inspired computing is a game changer for practical side channel research:

- Can work on asserting sound leakage bounds
- Have ability to produce scalable implementations:
  - Research perspective: to compute SR and GE curves and so explain the effectiveness of attack strategies across different leakage models, and SNRs
  - Practical perspective: To `emulate` the best real world attackers, to be used in evaluations & testing

All research done thanks to the University of Bristol HPC platform Blue Crystal.
Crypto Theory vs. Crypto Practice

Theory:
• A scheme is secure if a game is ‘hard’ to win
• (example above relates to symmetric encryption)

Practice:
• adversary also gets leakage
• (how do we include this in the theoretical game?)

O1: How to define and model leakage
O2: How to measure key entropy loss due to leakage
O3: How to build practical leakage resilient crypto