A Hacker’s guide to reducing side-channel attack surfaces using deep-learning

with the help of many Googlers and external collaborators
Talk is based on some of the results of a joint research project with many collaborators on hardening hardware cryptography.
Work in progress
Experimental results and code ahead
Side channel attacks are one of the most efficient ways to attack secure hardware.
A side-channel attack was used to recover the Trezor bitcoin wallet private key

https://jochen-hoenicke.de/crypto/trezor-power-analysis/
Side-channels attacks are notoriously **hard** to debug and fix.
Can we create a debugger that accurately pinpoints the code vulnerable to side-channel attacks?
Combine deep-learning and dynamic analysis to pinpoint origin of leakage
AI? Really?
Side Channel Attacks
Leak Detector
Today’s goal: use SCALD to debug tinyAES running on STM32F4
Agenda

- What are side channels?
- AI based side-channel attacks
- AI explainability
- Finding implementation leakage origin with SCALD
Code and slides
https://elie.net/scald
Disclaimer

This talk purposely focuses on showcasing a high level overview of how to debug a cryptographic implementation end-to-end using SCALD. For technical details, see the paper
Part 1
What are side-channel attacks?
A side-channel attack is an indirect measurement of a computation result via an auxiliary mechanism.
Real-world side-channel applications

- Recover encryption keys
- Perform blind SQL injections
- Steal passwords and pins
- Extract crypto wallets

Google
Security and Privacy Group
Data in

Plaintext

Secret Key

Leakage out

Timing

Current

Heat

Electromagnetic emission
AES round are visible in lightly protected AES implementation power traces
SCA in a nutshell

1. Encryption
2. Signal acquisition
3. Template attack
4. AES key!
NewAE Chipwhisperer Pro + Picoscope 6000 for fast sampling rate is what we use for our research

This is not an ad :) it is a recommendation based on what we use
Section 2
AI based side-channel attacks
Side Channel Attacks Automated with Machine Learning
How do SCAAML attacks work in practice?
Check out last year's talk for an in-depth explanation:

https://elie.net/scaaml
Threat model

whitebox attack

Contrary to our previous work that focused on black box attacks, the traces used in this talk are truncated and collected synchronously to improve debugging quality. This is consistent with the white-box attack model used during chip development. Additionally, the model architecture is also optimized for debugging, not pure performance.
SCAAML process overview

- Encryption
- Signal acquisition (ChipWhisperer)
- Predictions using DNN
- Combine DNN predictions
- AES key!
TinyAES has multiple attack points that can be targeted by SCAAML.

Today we focus on sub_bytes_in
Probabilistic attack: single trace

Val 0: 0.10
Val 1: 0.02
Val 2: 0.01
... 
Val 42: 0.3
... 
Val 254: 0.02
Val 255: 0.05
Probabilistic attack: summing predictions*

\[
\begin{align*}
\text{Val 0: 0.10} \\
\text{Val 1: 0.02} \\
\text{Val 2: 0.01} \\
\ldots \\
\text{Val 42: 0.3} \\
\ldots \\
\text{Val 254: 0.02} \\
\text{Val 255: 0.05}
\end{align*}
\]

\[
\begin{align*}
\text{Val 0: 0.08} \\
\text{Val 1: 0.04} \\
\text{Val 2: 0.05} \\
\ldots \\
\text{Val 42: 0.12} \\
\ldots \\
\text{Val 254: 0.03} \\
\text{Val 255: 0.10}
\end{align*}
\]

\[
\begin{align*}
\text{Val 0: 4.4} \\
\text{Val 1: 5.3} \\
\text{Val 2: 3.2} \\
\ldots \\
\text{Val 42: 21.4} \\
\ldots \\
\text{Val 254: 2.9} \\
\text{Val 255: 4.2}
\end{align*}
\]

\[+ \ldots +\]

*sum uses \(\log_{10} + \varepsilon\)
Model architecture
Hypertuned residual separated 1D convolution network

Custom residual block used
Tensorboards - 1 model per byte
Our side-channel optimized model architecture yield 16 high accuracy model in 5 epoch as expect on this easy use-case.
How to find where TinyAES is leaking using our model?
Section 3
Deep-learning explainability
A classic vision model prediction

Boxer

Tiger cat
Why did the model predict a tiger cat and a boxer?
Why did the model predict a tiger cat and a dog?

Explainability to the rescue: boxer prediction
Why did the model predict a tiger cat and a dog? Explainability to the rescue: cat prediction.

Google Security and Privacy Group
Identifying errors and biases

Horse-picture from Pascal VOC data set

- Source tag present
- Classified as horse
- No source tag present
- Not classified as horse
How do I use explainability and combine it with dynamic analysis to debug leakages?
Section 4
Finding leakage origin with SCALD
SCALD: Game plan

Model → Explainer → Leakage map → Target emulator (cpu + firmware) → Annotated code

Traces + predictions
Many explainability techniques exist.
Which explainability techniques work best?
Aggregate, filter, and normalize

Reduce to key spikes
Byte 0 leak map visualization for various techniques

SNR
Grad Cam++
Activations maps
Benchmarking key explainability techniques

Test traces → mask top n points → model → Accuracy decrease

Leak map
Benchmark results: lower is better

<table>
<thead>
<tr>
<th></th>
<th>Byte 0</th>
<th>Byte 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>SNR</td>
<td>57%</td>
<td>44%</td>
</tr>
<tr>
<td>Activation maps</td>
<td>58%</td>
<td>95%</td>
</tr>
<tr>
<td>Grad Cam++</td>
<td>58%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Preliminary results - 4 points masked
Explainability techniques don’t work better than SNR and have very noisy leak maps.
Develop a technique tailored to leakage explanation
Custom code? Really?
SCALD explainer combines partitioned and convolutive occlusion for speed and precise leakage pinpointing.
Benchmark results: lower is better

- **Baseline**: 100%
- **SNR**:
  - Byte 0: 57%
  - Byte 7: 44%
- **Activation maps**:
  - Byte 0: 58%
  - Byte 7: 95%
- **Grad Cam++**:
  - Byte 0: 58%
  - Byte 7: 95%
- **SCALD**:
  - 17%

Preliminary results - 4 points masked
byte 0 leak maps comparison: the SCALD map is visibly cleaner
SCALD custom explainability technique decreases accuracy the most and generate low noise leak map.
How do you go from the leakage map to code?
From traces to CPU instructions

state automatons

FW

CPU

Leakage map

Mapped ASM

Security and Privacy Group
From CPU instructions to code

- Firmware
- Debug symbol
- Mapped ASM
- Code mapper
- Code leakage mapping
Theory looks great but how hard is it in practice?
Requirements

An explanation technique that have single point precision
We need to isolate the exact few points of the traces that cause most of the leakage as some instruction only take one cycle or two (4 or 8 traces points)

An emulator that have single cycle precision
We need to map each instruction to its exact cycle to be able to map them to the trace. A single error and the entire analysis is wrong as all instruction will be shifted.

A bit of computation
You need a 1M data point dataset, 16 models, 16 explanations, 1 full target execution and 1 mapping. With all our optimization this is requires a few days of computation that are parallelizable.
This level of explainability and emulation precision seems out-of-reach.
Model targeting `sub_bytes_in` are expected to mostly exploit leakage in the `AddRoundKey()` function.
TinyAES aes.c line 213 is exactly the sub_byte_in operation! SCALD perfectly identify the main source of leakage.
SCALD is able to automatically isolate the exact code vulnerable to a given SCAAML side-channel attack.
SCALD annotated code empowers developers to quickly figure out what to patch and focus on developing stronger crypto.
Takeaways

- SCAAML attacks allows to perform SOTA SCA attacks automatically
- SCALD use AI to find automatically leakage origin - reducing development cost
- AI for side-channel is still a nascent field with a lot of exciting opportunities
Keep up with our research on deep-learning for side-channel attacks: https://elie.net/scaaml