Federal Office
for Information Security

Breaking Masked Implementations of Clyde by Means of Side-Channel Analysis

Aron Gohr [1] Friederike Laus [2] Werner Schindler [2]

[1] Independent Researcher

[2] Federal Office for Information Security

September, 21th 2022

Outline

- Side-Channel-Analysis: An Introduction
- CHES Challenge 2020: Task Description
- Blueprint for a Solution
- Minor and Major Problems
- The Scattershot Encoding or How We Learned to Stop Worrying About Convergence of Single Target Variables
- Further Analysis: Analyzing the Scattershot Encoding on a Synthetic Problem, Comparison to Stochastic Approach

Side-Channel-Analysis

Background: Side-Channel Analysis

- Mathematical cryptanalysis tends to be difficult with unweakened versions of modern ciphers.
- Much easier to break the implementation!
- One way to do this is by side-channel attacks: use physical side effects of computation to gain additional information.
- Side-channel attacks are a very practical threat when the adversary can do the required measurements.

Side-Channel Analysis: Countermeasures

- Can we do anything about it?
- Yes; various things:
  - Drown the signal in artificially generated noise.
  - Put the device in a Faraday's cage in a bunker surrounded by trusted armed guards.
  - Make alignment difficult by introducing jitter.
  - Run dummy operations.
  - Employ logical masking.
- *Masking* has the best theoretical justification of these methods, but tends also to be relatively expensive.
- Therefore interest in *masking-friendly* ciphers.

Masking: Basic Ideas

- Usually, a side-channel adversary does not get enough information about the secret from a *single* run of the device to recover much of it.
- Therefore, they will combine information across traces in the hopes of reconstructing the full secret: it remains constant from one trace to the next.
- Basic idea of masking:
  1. Break up the secret key and the data being processed into shares in a secret-sharing scheme.
  2. Run the encryption on the shares.
  3. Refresh the shares across executions or even within one execution of the scheme.
- This has two effects:
  1. The bit-representation of the secret becomes longer, i.e. more bits the adversary has to recover in single-trace attacks.
  2. The secret-shared state changes after every refreshing of the shares, making it difficult to combine information across traces.

CHES Challenge 2020

CHES Challenge 2020: The Task I

- Challenge: break masked implementations of the masking-friendly *Clyde* cipher by power analysis.
- Tweakable lightweight block cipher, 128 bit block, key and tweak size.
- Seven different targets, four software and three hardware targets.
- All of them masked implementations; software targets had 3,4,6,8-fold masking.
- Only the software targets were solved.
- Power traces for the software targets were captured at fairly high resolution (62.500 to 218.750 data points per trace, depending on the challenge).
- Power traces included roughly the first round of the cipher, but *excluded* randomness generation; hence, randomness used for share generation/refreshment had to be treated as completely random.

CHES Challenge 2020: The Task II

- *Breaking* a challenge meant achieving a mean key rank $< 2^{32}$ using a chosen number of traces.
- Goal of the contest: break as many challenges as possible with as few traces as possible.
- It turns out we were the only group to submit a solution to any of the challenges.

CHES Challenge 2020: The Task III

- Data given during training: 200.000 power traces together with secret keys, secret key shares, tweak values, plaintext values (all zero) for each challenge.
- A Python implementation of the masked Clyde logic was also supplied by the organizers.
- Attacks could be tested against up to 100.000 traces for a few fixed keys for each target.
- Entrants were evaluated against a data set kept secret by the organizers. Key ranking by standard histogram based methods.

A Brief Introduction to the Clyde Cipher

- Acts on blocks of 128 bits, arranged as 4×32 bit array $x \in \mathbb{F}_2^{4\times 32}$
- Consists of 6 rounds, each round is composed of
  - tweakey addition
  - S-box (acts on columns)
  - L-box (acts on rows)
  - addition of a constant
- Another tweakey addition happens in the final round.
- A fairly simple tweakey schedule is used to mix key and tweak.
- Implementable with a fairly small number of AND and OR gates.

A Solution Using Deep Neural Networks

Code and data: `https://github.com/agohr/ches_ctf_2020`

Goals

We wanted to achieve the following:

- If possible win the contest by having the most efficient attacks.
- Use techniques that do not need too much manual tuning, i.e. hyperparameter, point of interest or leakage target selection.
- Break all the software challenges using the same methods.

Blueprint of a Solution

- Get predictions for all shares (with some uncertainty).
- Derive a guess for the unshared state (accepting increase in uncertainty).
- The unshared state depends only on Clyde logic and inputs, not on masking.
- For each key hypothesis, calculate the unshared state given all inputs to Clyde and compare the results to the results of side-channel extraction.

Minor Problems

- How to pick a leakage target?
  - We target the state after the first S-box; this is a fairly standard choice.
- How to avoid a large search cost when ranking key hypotheses?
  - Given the leakage target, this is easy: we can rank one nibble at a time, i.e. only 16 hypotheses per nibble.
- Can we process large traces without manual POI selection?
  - We reuse a neural network structure introduced at SAC 2020 by Gohr, Jacob and Schindler which was designed to handle large traces directly.
  - Main ideas: deal with large trace sizes by subsampling with varying offset and combining predictions made for the subtraces; deal with class imbalance by framing prediction as regression instead of classification.

Major problems

- When we tried to implement this, our networks saw many bits of the shared state quite well, but *failed to converge* for others.
- Each bit in the shared state that we just *don't see* means that there is one bit in the unshared state that we just *don't see*.
- When we do not see any biases for a bit of the state, *more data is not going to resolve the problem*.

Obtaining Convergence: The Scattershot Encoding

Problem:

- When predicting the internal cipher state naively, all predicted bits are close to independent of each other.
- This means we can make progress on some bits while not learning how to predict the others.

Possible solution using the Scattershot Encoding:

- Pick *random subsets* $S_1, S_2, \ldots, S_n$ of the target bits and predict $hw(S_i)$.
- Given noisy predictions of these values, we can obtain noisy predictions of all the target bits by linear algebra.
- Subproblems are now all strongly related to each other (S_i not disjoint), so convergence should be more uniform.

Results I

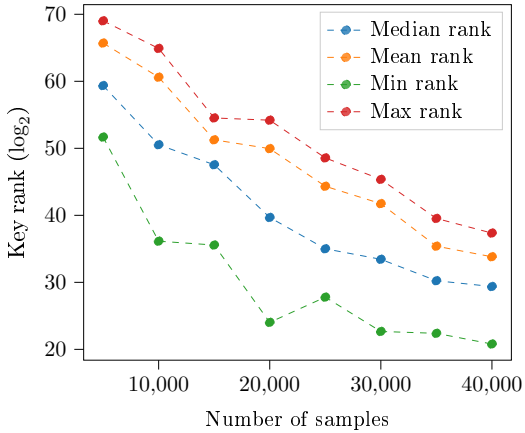


Figure: Sw8 challenge

Results II

Number of traces required to obtain median rank $< 2^{32}$:

- Sw3: ≈ 25
- Sw4: ≈ 105
- Sw6: ≈ 3.000
- Sw8: ≈ 35.000

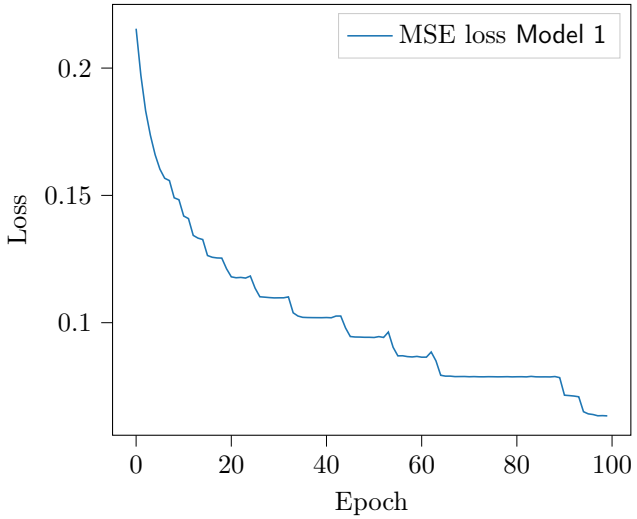
After the challenge was closed, another solution was published by Bronchain and Standaert based on deep belief networks. Their solution is more efficient than ours for SW6/SW8, comparable for SW4, and less efficient for SW3.

Further Analysis

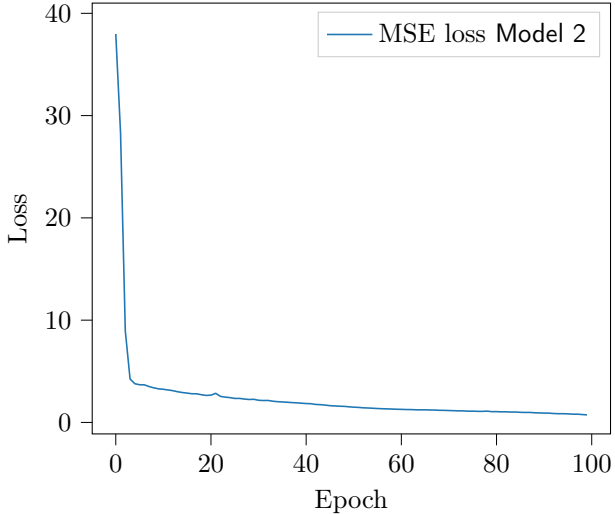
Dissecting the Scattershot Encoding

- The Scattershot Encoding just multiplies the values we would *like* to predict by a fixed *random binary matrix* before running training.
- Why does this help, again?
- To answer this, we designed a simple synthetic problem: learn to predict the function $F(x_1, x_2, \ldots, x_n) := (x_1, x_1 \oplus x_2, \ldots, \oplus_{i=1}^{n} x_i)$.
- No noise, no randomness, a simple function to predict, so this is easy, right?
- Higher components are *difficult* to learn for a neural network.
- Naive approach and scattershot approach show qualitatively similar behaviour on this synthetic problem as on the side-channel challenge.

Synthetic Problem: Naive Approach



Synthetic Problem: Scattershot Encoding



Comparison With the Stochastic Approach

- We also tried to solve the challenge with a “classical”, statistical approach, namely the stochastic approach, which required adapting the stochastic approach to dealing with secret-shared keys.
- Accounting for the high number of masking bits becomes quickly intractable, even for the Sw3-Challenge.
- Both the scattershot encoding and the stochastic approach see large biases in the same *S*-boxes, however.
- We found that our neural networks draw in information from some parts of the recorded traces that are surprising and therefore missed by the manual POI analysis done for the stochastic approach experiments.
- Overall, our neural network based solution significantly outperforms the stochastic approach.

Advantages and Disadvantages of the Two Approaches

- The deep learning approach has a greater capacity to find complex unanticipated features and sources of leakage; for the task here under consideration, it therefore yields better results.
- However, the features exploited by the stochastic approach are human-comprehensible by design.
- Our work on the Scattershot Encoding reinforces the known point that failure of a neural network to *find* a solution does not imply that a solution does not exist or even that a solution cannot be represented by the neural network; in this regard, traditional approaches are probably more *sound* in the sense that if a good model exists in the model family considered, it will be found.

Conclusion and Future Work

Conclusion

- Implementations with significant protection can still be attacked by neural networks without a need for the analyst to perform a deep analysis of the target.
- However, masking works in principle even in a relatively low-noise setting; the number of traces required for a successful attack rises quickly with masking order.
- Study of simple, synthetic problems can be quite insightful for deep learning based side-channel analysis.

Future Work

- A deeper understanding of the Scattershot Encoding would be desirable. Are there other similar tricks?
- Learning loss functions for side-channel analysis?

Thanks for your attention!

Contact

Aron Gohr
aron.gohr@gmail.com